ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIERO INDUSTRIAL

DISEÑO DE UNA TARJETA DE ADQUISICIÓN DE DATOS DE 12 CANALES SIMULTÁNEOS

Autor: Juan Guisández Méndez
Director: José Daniel Muñoz Frías

Madrid
2011

DISEÑO DE UNA TARJETA DE ADQUISICIÓN DE DATOS DE 12 CANALES SIMULTÁNEOS

Autor: Guisández Méndez, Juan.

Director: Muñoz Frías, José Daniel.

Entidad colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

1. Introducción.

En el mercado actual no existen tarjetas de conversión simultánea con los suficientes canales, resolución y velocidad para satisfacer los requisitos de este proyecto.

El uso de una conversión simultanea de todas las variables es de vital importancia para evitar errores en el algoritmo de control y por eso es necesario este proyecto.

En la mayoría de procesos ingenieriles es vital la toma de datos de múltiples variables simultáneas analógicas, puesto que los sistemas informáticos son una herramienta fundamental en la toma de decisión a partir de los datos obtenidos, es necesario desarrollar un modo de convertir las señales analógicas propias del proceso en señales digitales que el ordenador pueda interpretar y manejar. En el caso concreto de este conversor su uso será en el control de un motor eléctrico.

2. Motivación.

El objetivo de este proyecto es la realización del diseño de una tarjeta de conversión analógico/digital, capaz de muestrear doce canales analógicos simultáneos con una resolución de 16 bits y una velocidad de muestreo de al menos 250ksps. Dicha tarjeta se conecta al bus PCI del ordenador.

3. Desarrollo.

La realización de este proyecto se divide principalmente en tres partes. Primero se diseñó el esquema eléctrico correspondiente a los conversores, luego se diseñó el PCB donde quedarían alojados y por último se diseñó el circuito VHDL que se encarga de gestionar la información y de controlar los conversores.

   a. Diseño eléctrico.

El primer paso para el diseño eléctrico fue decidir los conversores que se iban a utilizar así como el resto de componentes electrónicos. Se decidió utilizar dos conversores de seis canales cada uno para alcanzar el objetivo de los doce canales. El resto de los componentes fueron elegidos teniendo en cuenta la nota de aplicación de los conversores.

También se añadió una etapa amplificadora de ganancia variable, mecánicamente mediante el uso de un jumper, para aumentar las prestaciones del equipo y su rango de operación.

b. Diseño del PCB.

Debido al espacio disponible, a que los conectores que tendrían que ir conectados en la FPGA eran puntos fijos, y la complejidad del diseño, se decidió utilizar un PCB de 4 capas. La cara superior se utiliza para poner los componentes y gran parte de las pistas, la cara inferior se utiliza para poner el resto de las pistas. Las capas internas se usan, una como capa de tierra, separando convenientemente la tierra analógica de la tierra digital para evitar interferencias. La otra capa se usa como capa de tensión que sirve de alimentación para conversores y operacionales.

Puesto que las patillas del conversor están muy cerca, nos hemos visto obligados a utilizar una clase de PCB alta, lo que nos permite usar pistas y separaciones entre pistas pequeñas pero que incrementa el precio de producción.

c. Diseño de la FPGA.

La tarjeta FPGA utilizada es la Raggedstone1 Spartan3. La principal razón de utilización de esta tarjeta es que ya se disponía de ella y cumple perfectamente las necesidades de este proyecto.

Para el diseño del circuito de la FPGA se utilizo un diseño modular, se diseñaron varios componentes más o menos complejos y luego se unieron para conformar el circuito final.

Podemos diferenciar entre dos tipos de componentes, unos sencillos y que incluso se utilizan en múltiples ocasiones en el circuito final como pueden ser los registros o las puertas triestado. Por otro lado tendríamos el componente que hemos llamado controlador que es mucho más complejo y que se encarga de gestionar la interacción con los conversores.

Para la conexión con el puerto PCI se utiliza un core gratuito proporcionado por opencores que realiza el interfaz con el PCI y para comunicarse con el core, se usa el bus wishbone que es un bus estándar y ampliamente extendido. Esto además permite que sea fácil de adaptar el sistema a otro tipo de puertos o de sistemas utilizando el bus wishbone cambiando muy poco del circuito FPGA.

4. Resultados.

Hemos conseguido diseñar una tarjeta de conversión con unas muy buenas características de velocidad, resolución y número de canales y especialmente por el hecho de realizar estas conversiones de manera simultánea, lo cual es muy difícil de encontrar en el mercado, y aun así el precio de producción para cantidades medias altas es más que aceptable.

5. Conclusiones.

Se ha conseguido diseñar una tarjeta de doce canales de muestro simultáneos a un precio asequible la cual dispone de una gran flexibilidad debido al uso de una FPGA para su conexión al PC mediante el puerto PCI.

Se han realizado múltiples simulaciones para validar el funcionamiento tanto de los componentes por separado como del conjunto obteniéndose buenos resultados. Todos los objetivos propuestos para el proyecto han sido alcanzados, aunque por falta de presupuesto no se ha podido fabricar y probar el prototipo.

DESIGN OF A 12 CHANNEL SIMULTANEOUS DATA ACQUISITION CARD

Author: Guisández Méndez, Juan.

Director: Muñoz Frías, José Daniel.

Sponshorship: ICAI – Universidad Pontificia Comillas

PROYECT SUMMARY

1. Introduction.

In today's market there are no cards of simultaneous conversion with enough channels, resolution and speed to meet the requirements of this project.

The use of a conversion simultaneously of all the variables is of vital importance to avoid mistakes in the process of control.

In the majority of engineering processes is vital to taking data from multiple simultaneous analog variables, since that computer systems are a fundamental tool in the decision-making process from the data obtained, it is necessary to develop a way of turning own process analog signals into digital signals that the computer can interpret and handle. In the case of this converter, their use will be in control of an electric motor.

2. Motivation.

The objective of this project is the design of a card of converting analog/digital, able to sample twelve simultaneous analog channels with a resolution of 16 bits and a sampling of at least 250ksps rate. This card will have to be able to communicate with a computer via a PCI bus.

3. Development.

The realization of this project is mainly divided into three parts. First design the electrical scheme corresponding to the converters, then design the PCB where would be hosted and finally design VHDL circuit which is responsible for managing information and to control converters.

   a. Electric design.

The first step for the electrical design was to decide the converters that were to be used as well as other electronic components. We were decided to use two converters of six channels each one to achieve the goal of the twelve channels. The rest of the

components were chosen taking into account the form of implementation of the converters.

Also we added an amplifier stage of variable gain, mechanically through the use of a jumper, to increase the performance of the team and its range of operation.

b. Design of the PCB.

Due to the space available, the connectors, that would have to go online in the FPGA, were fixed points, and the complexity of the design, it was decided to use a 4 layers PCB. The upper side is used to put the components and most of the tracks, the underside is used to put the rest of the tracks. The inner layers are used, one layer of ground, properly separating the analog ground of digital ground to avoid interference. The other layer is used as a layer of voltage which supply power to converters and operational.

Since the pins of the converter are very close, we have been forced to use a high class of PCB, allowing us to use tracks and separations between small tracks but that increases the price of production.

c. Design of the FPGA.

The FPGA card used is the Raggedstone1 Spartan3. The main reason for use of this card is that already there were it and perfectly meets the needs of this project.

For the FPGA circuit design using a modular design, designed several, more or less complex, components and then joined to form the final circuit.

We can differentiate between two types of components, some simple and even used multiple times in the final circuit such as registers or the TriState doors. On the other hand would have the component, that we call controller, that is much more complex and which is responsible for managing the interaction with the converters.

It is used for the connection to the PCI port a core free provided by opencores which makes the interface with the PCI and to communicate with the core, it uses the bus wishbone which is a standard and widely extended bus. This also allows to be easy to adapt the system to other types of ports or systems using the wishbone bus, changing very little of the FPGA circuit.

4. Results and conclusions.

We have managed to design a card of conversion with very good characteristics of speed, resolution and number of channels and especially by the fact of these

conversions in a way simultaneous, which is very difficult to find in the market, and still the price of production for quantities average or high is more than acceptable.

5. Conclusions.

He has been designing a card of twelve channels of simultaneous sampling at an affordable price which provides great flexibility due to the use of an FPGA for their connection to the PC via the PCI port.

However there have been multiple simulations to validate the operation, both of the components separately as all together, getting good results. All the objectives proposed for the project have been achieved, but due to lack of budget not been able to manufacture and test the prototype.

Índice de la memoria

Parte I Memoria ***7***

Capítulo 1 Introducción ***8***

1.1 Estudio de los trabajos existentes / tecnologías existentes **8**

1.2 Motivación del proyecto **9**

1.3 Objetivos **9**

1.4 Metodología / Solución desarrollada **10**

1.5 Recursos / herramientas empleadas **10**

Capítulo 2 Desarrollo técnico ***11***

2.1 Elección de los elementos a utilizar **11**

2.1.1 Conversor analógico digital ADS8556 de Texas Instruments 11

2.1.2 Raggedstone1 Spartan-3 FPGA PCI Development Board 14

2.1.3 Operacionales OPA2211a 14

2.1.4 Resistencias, condensadores y otros componentes 15

2.2 Diseño del circuito eléctrico **16**

2.2.1 Adaptación de las señales a muestrear 16

2.2.2 Condensadores para acondicionar el conversor 17

2.2.3 Conexión con la FPGA 18

2.2.4 Conexiones del conversor 20

2.3 Diseño del PCB **21**

2.4 Diseño de la FPGA **22**

2.4.1 PCI32TLITE 24

2.4.2 Bus Wishbone 26

2.4.2.1 Introducción 26

2.4.2.2 Señales utilizadas 27

2.4.2.3 Operaciones generales. 28
2.4.2.3.1 Ciclo de lectura (Figura 14) 28
2.4.2.3.2 Ciclo de escritura (Figura 15) 30
2.4.3 Controlador 32
2.4.3.1 Introducción 32
2.4.3.2 Explicación de los registros 33
2.4.3.2.1 Registros tipo CR X.Y 33
2.4.3.2.2 Registros tipo ZXY 36
2.4.3.2.3 Registro TIMER 36
2.4.3.2.4 Registro CONTROL 36
2.4.3.2.5 Registro BUSY 37
2.4.3.3 Máquinas de estado. 37
2.4.3.3.1 Máquina de estados principal. 37
2.4.3.3.2 Máquina de estados de la inicialización. 38
2.4.3.3.3 Maquina de estados de la conversión. 40

Capítulo 3 Resultados/Experimentos 42

Capítulo 4 Conclusiones 51

Capítulo 5 Futuros desarrollos 52

Bibliografía 53

Parte II Estudio económico 54

Capítulo 1 Estudio económico 55

Parte III Manual de usuario 56

Capítulo 1 Manual de usuario 57

1.1 Instalación del hardware. 57

1.2 Desarrollo del software. 57

1.2.1 Uso de los registros. 57
1.2.2 Procedimiento a seguir. 59

Parte IV Código fuente 60

Capítulo 1 Tarjeta 62

1.1 Registro. 62

1.2 Triestado..... 63

1.3 Multiplexor2..... 64

1.4 Multiplexor13..... 65

1.5 Demultiplexor6 66

1.6 Controlador..... 67

1.7 Observador..... 73

Capítulo 2 Simulaciones..... 76

2.1 pruRegistro 76

2.2 pruTriestado 78

2.3 pruMul2..... 80

2.4 pruMul13..... 82

2.5 pruDemul6 85

2.6 pruControl 87

2.7 pruObservador 92

2.8 Conjunto..... 94

2.9 Final..... 106

Parte V Datasheets..... 118

Capítulo 1 ads8556 120

Capítulo 2 opa2211..... 122

Índice de figuras

Figura 1: ADS8556 13
Figura 2: Raggedstone1 Spartan-3 FPGA Development Board..... 14
Figura 3: Adaptación de las señales 16
Figura 4: Condensadores del conversor 17
Figura 5: Conexión datos del conversor..... 18
Figura 6: Conexión configuración del conversor 19
Figura 7: Conexiones del conversor..... 20
Figura 8: Distancias en el PCB..... 22
Figura 9: Diagrama de bloques 23
Figura 10: Configuración inicial 24
Figura 11: Ciclo de lectura simple bus PCI..... 25
Figura 12: Ciclo de escritura simple bus PCI..... 26
Figura 13: Bus wishbone..... 28
Figura 14: Ciclo de lectura 30
Figura 15: Ciclo de escritura 32
Figura 16: Esquema lógico..... 33
Figura 17: Maquina de estados principal 37
Figura 18: Maquina de estados de la inicialización 39
Figura 19: Maquina de estados de la conversión 40
Figura 20 : Carga conversores..... 43
Figura 21: Carga tiempo de ejecución..... 44

Figura 22: Inicio conversión simple. 45
Figura 23: Configuración de los conversores 46
Figura 24: Tiempo de conversión. 47
Figura 25: Lectura de los conversores 48
Figura 26: Fin de la conversión 49
Figura 27: Lectura de los registros. 50

Índice de tablas

Tabla 1: Tarjetas de conversión 8
Tabla 2: Comparativa de conversores 11
Tabla 3: Enob de los conversores 12
Tabla 4: Relación consumo velocidad 13
Tabla 5: Diseño de PCBs 21
Tabla 6: Registro CR X.1 34
Tabla 7: Registro CR X.2 34
Tabla 8: Registro XYZ 36
Tabla 9: Registro TIMER 36
Tabla 10: Registro CONTROL 36
Tabla 11: Registro BUSY 37
Tabla 12: Registros de escritura 58
Tabla 13: Registros de lectura 59

Parte I MEMORIA

Capítulo 1 INTRODUCCIÓN

1.1 ESTUDIO DE LOS TRABAJOS EXISTENTES / TECNOLOGÍAS EXISTENTES

Existen muchas tarjetas PCI de conversión analógica digital disponibles en el mercado, sin embargo es más difícil encontrar que estas tarjetas permitan la conversión simultánea de las variables que muestrean. Podemos encontrar distintas tarjetas como las que se muestran a continuación (Tabla 1).

Modelo	*Nº de canales*	*Resolución*	*Velocidad*	*Precio*
PCI 4472B	8	24 bits	102,4kpsps	3849€
PCI 6143	8	16 bits	250ksps	1199€
PCI 7833R	8	16 bits	200ksps	3399€
PCI 7831R	8	16 bits	200ksps	2599€
PCI 6123	8	16 bits	500ksps	2549€

Tabla 1: Tarjetas de conversión

Teniendo en cuenta los resultados obtenidos se pueden sacar las siguientes conclusiones.

Primero, la resolución no es un problema, es fácil encontrar tarjetas de conversión analógica digital con una resolución de 16 bits o más. Aunque es fácil encontrar tarjetas con una velocidad de 250ksps o incluso mayores, el número de tarjetas que permiten esa velocidad se reduce drásticamente.

El problema llega al buscar el número de canales analógicos que se pueden muestrear simultáneamente, ya que solo se han encontrado tarjetas PCI capaces de convertir 8 canales de manera simultánea, lo cual es insuficiente para nuestras

necesidades ya que queremos tener la capacidad de muestrear 12 canales simultáneos.

Llegados a este punto la solución podría ser la utilización de dos tarjetas para el muestreo de las señales, pero esto supondría el uso de dos tarjetas PCI y por lo tanto un mayor consumo de los recursos disponibles. Además las tarjetas disponibles en el mercado son muy caras.

Por todo esto se decide hacer una tarjeta de conversión analógico digital que se conecte a un único puerto PCI que sea capaz de convertir 12 canales simultáneos y con una resolución de 16 bits e intentando en la medida de lo posible hacer un diseño lo más económico posible.

1.2 MOTIVACIÓN DEL PROYECTO

En la mayoría de procesos ingenieriles es vital la toma de datos de múltiples variables simultáneas analógicas, puesto que los sistemas informáticos son una herramienta fundamental en la toma de decisión a partir de los datos obtenidos, es necesario desarrollar un modo de convertir las señales analógicas propias del proceso en señales digitales que el ordenador pueda interpretar y manejar. El uso de una conversión simultánea de todas las variables es de vital importancia para evitar errores en el proceso de control. En el caso concreto de este conversor su uso será en el control de un motor eléctrico.

1.3 OBJETIVOS

Desarrollo de una tarjeta de conversión analógico/digital, capaz de muestrear doce canales analógicos simultáneos con una resolución de 16 bits y una velocidad de muestreo de al menos 250ksps. Dicha tarjeta tendrá que poder comunicarse con un ordenador mediante un bus PCI.

1.4 METODOLOGÍA / SOLUCIÓN DESARROLLADA

Para realizar el objetivo propuesto se han llevado a cabo los siguientes pasos:

Se decidió la tarjeta FPGA así como el conversor analógico digital que se utilizan. Este paso se realizó eligiéndose la tarjeta Raggedstone1 Spartan-3 FPGA PCI Development Board y dos conversores analógico digital ADS8556 de Texas Instruments.

Después de tener los elementos de hardware principales decididos se diseñó el circuito eléctrico necesario para el conversor, la adaptación de las señales a muestrear y la conexión con la tarjeta FPGA.

Tras lo cual se diseñó el pcb para el circuito antes diseñado y que se conectara á la tarjeta FPGA.

Hecho esto mediante el uso de código VHDL se diseñará el circuito que se descargará en la tarjeta FPGA para realizar el interfaz entre conversores y el pc para el correcto tratamiento de los datos aportados por el conversor.

1.5 RECURSOS / HERRAMIENTAS EMPLEADAS

Se ha utilizado, a parte de la propia tarjeta y el conversor, aquellos elementos necesarios para implantar el sistema eléctrico (operacionales, resistencias, condensadores).

Además se utilizaron diversos paquetes de software entre los que se incluye, Xilinx ISE Design Suite 11 y 12 en su versión gratuita, para el diseño del circuito a configurar en la FPGA del código VHDL.

Así como los programas de software libre gEDA, para el diseño del circuito eléctrico y el programa también de software libre PCB, para el diseño del pcb ambos ejecutados bajo Linux.

Como programa de tratamiento de textos se utilizó Microsoft Word 2007.

Capítulo 2 DESARROLLO TÉCNICO

2.1 ELECCIÓN DE LOS ELEMENTOS A UTILIZAR

En los siguientes apartados se procede a realizar la justificación de la utilización de los distintos componentes utilizados en el diseño de la tarjeta.

2.1.1 CONVERSOR ANALÓGICO DIGITAL ADS8556 DE TEXAS INSTRUMENTS.

Los requisitos mínimos necesarios que tiene que tener el conversor de forma de que sea adecuado para el proyecto son: una velocidad de muestreo de al menos 250ksps, una resolución de 16 bits y el muestreo de 6 canales simultáneos. La búsqueda realizada entre los proveedores de Texas Instruments y Analog Devices arrojó los resultados que se muestran (Tabla 2).

Modelo	*Proveedor*	*Nº Bits*	*Velocidad*	*SINAD*	*Consumo*	*Paralelo*	*Serie*	*Precio*
ADS8364	Texas Ins.	16	250ksps	82.5	413.0mw	Si	No	18.10$
ADS8365	Texas Ins.	16	250ksps	87.0	190.0mw	Si	No	16.25$
ADS8556	Texas Ins.	16	630ksps	89.5	251.7mw	Si	Si	15.95$
AD7656-1	Analog De.	16	250ksps	87.8	140.0mw	Si	Si	17.95$
AD7656	Analog De.	16	250ksps	85.4	143.0mw	Si	Si	17.20$

Tabla 2: Comparativa de conversores

Con estos datos se calcula el Enob (effective numbers of bits, número efectivo de bits) a partir del SINAD (Signal-to-Noise Ratio plus Distortion, ratio señal ruido más distorsión) según la fórmula (E. 1)

$$Enob = \frac{SINAD - 1.76}{6.02}$$

E. 1

Modelo	*Enob*
ADS8364	13.41
ADS8365	14.16
ADS8556	14.90
AD7656-1	14.30
AD7656	13.90

Tabla 3: Enob de los conversores

A la vista de estos datos obtenidos se decide elegir el modelo ADS8556 de Texas Instruments (Figura 1) por los siguientes motivos.

_Es el modelo que permite un muestreo más rápido alcanzando una velocidad de 630ksps en interfaz paralela, aunque hay que tener en cuenta que con interfaz serie la velocidad baja hasta las 450ksps pero aun así sigue siendo el modelo más rápido incluso comparado con otros en interfaz serie.

_Es el modelo con un mayor Enob (Tabla 3) lo que significa una mayor precisión en las muestras obtenidas.

_Permite la utilización tanto de una interfaz serie como paralela.

_Es el modelo más económico, con un precio inferior a los 16 dólares americanos

El único punto donde el modelo elegido se ve superado por sus competidores es en el consumo energético, pero puesto que el consumo no es muy grande todas las ventajas adicionales, previamente expuestas de este modelo

compensan con creces este mayor consumo energético. Además si se calcula el consumo por ksps se obtiene:

Modelo	*Consumo/Velocidad*
ADS8364	1.652mw/ksps
ADS8365	0.760mw/ksps
ADS8556	0.399mw/ksps
AD7656-1	0.560mw/ksps
AD7656	0.572mw/ksps

Tabla 4: Relación consumo velocidad

Observando los resultados (Tabla 4) podemos ver que el modelo elegido es también el de menor consumo en cuanto a consumo por ksps.

Figura 1: ADS8556

2.1.2 RAGGEDSTONE1 SPARTAN-3 FPGA PCI DEVELOPMENT BOARD

La tarjeta Raggedstone1 Spartan-3 FPGA PCI Development Board (Figura 2) fue elegida más por razones prácticas que científicas, puesto que ya se disponía de la misma se puso como requisito para el proyecto el realizarlo usando esta tarjeta.

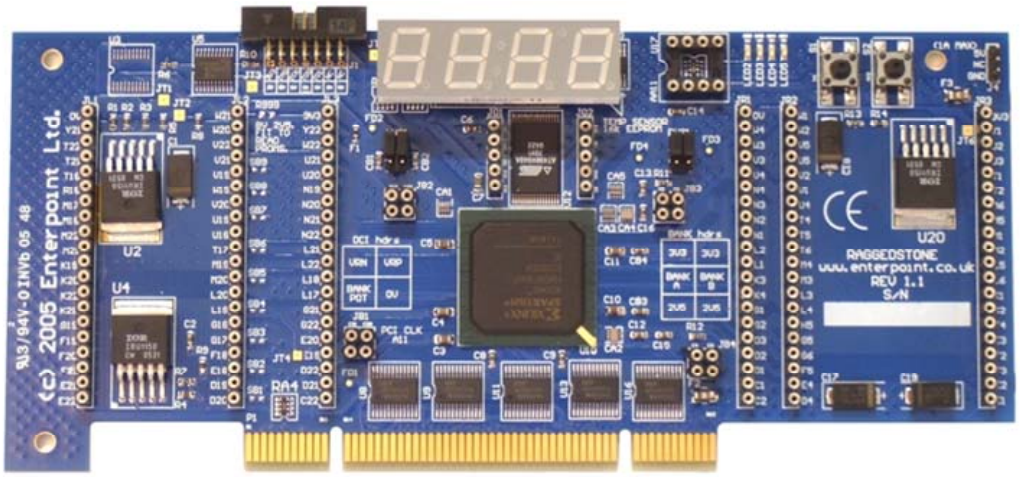


Figura 2: Raggedstone1 Spartan-3 FPGA Development Board

2.1.3 OPERACIONALES OPA2211A

Se ha elegido el operacional de dos canales OPA2211a de Texas Instruments principalmente por ser el operacional recomendado por el datasheet del conversor analógico digital, por lo que sabemos que van a funcionar bien conjuntamente.

2.1.4 RESISTENCIAS, CONDENSADORES Y OTROS COMPONENTES

Para la elección de los distintos condensadores y resistencias necesarios para la construcción de la tarjeta se han elegido del catalogo digital de la página web de farnell: http://es.farnell.com/

Se han tenido en cuenta diferentes criterios para la elección de los componentes:

1º Tamaño: se han elegido componentes de pequeño tamaño para hacer más sencillas las etapas posteriores pero respetando un tamaño mínimo que pudiese ser soldado con los medios a nuestra disposición.

2º Calidad: Los componentes elegidos se han elegido de la mayor calidad posible manteniendo un compromiso económico de elegir componentes baratos.

3º Precio y tamaño del pedido: Se ha tenido en cuenta el número de unidades que era necesario pedir, así como el número mínimo de unidades que hay que pedir y el precio asociado, sin olvidar incluir los costes asociados al transporte.

Con estos criterios se han elegido los siguientes componentes para la realización del proyecto:

_Resistencias de 1kΩ: CRCW06031K00FKEA de VISHAY DRALORIC.

_Resistencia de 22Ω: MFR3 22R FI de WELWYN

_Jumper: 0-0142270-3 de TYCO ELECTRONICS / AMP

_Condensador de 100nF: 06033C104JAT2A de AVX

_Condensador de 1µF: 0603YD105KAT2A de AVX

_Condensador de 10µF: 1206YC106KAT2A de AVX

_Condensador de 470nF: 0603YD474KAT2A de AVX

_Condensador de 1nF: 08052C102KAT2A de AVX

2.2 DISEÑO DEL CIRCUITO ELÉCTRICO

En el circuito eléctrico se pueden considerar distintas partes que se pueden diseñar de forma prácticamente independiente.

2.2.1 ADAPTACIÓN DE LAS SEÑALES A MUESTREAR

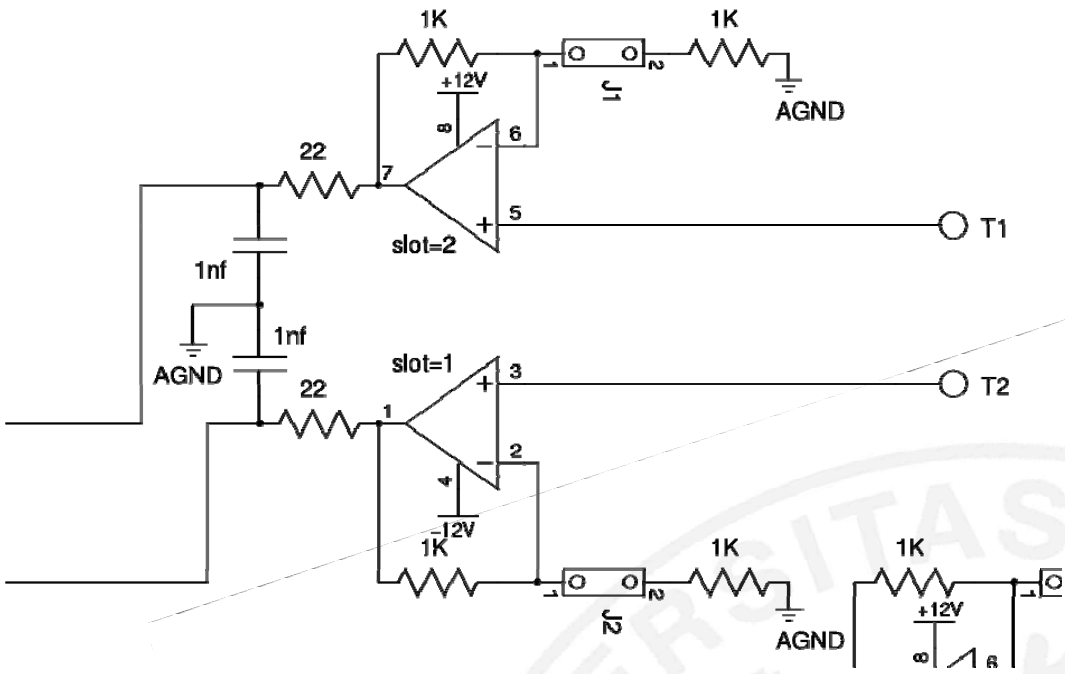


Figura 3: Adaptación de las señales

El circuito de las señales a muestrear (Figura 3) se ha diseñado uno para cada dos señales a muestrear, el circuito estará formado por cuatro resistencias de 1kΩ, dos resistencias de 22Ω, dos condensadores de 1nF, un operacional OPA2211a y dos jumper.

El sistema está diseñado para trabajar adecuadamente con tensiones comprendidas entre menos doce voltios y doce voltios. La inclusión en el circuito de la segunda resistencia de 1kΩ, que puedo o no ser conectada gracias al jumper,

permite variar la ganancia entre uno y dos de esta etapa para poder tener una precisión mayor para señales pequeñas.

2.2.2 CONDENSADORES PARA ACONDICIONAR EL CONVERSOR

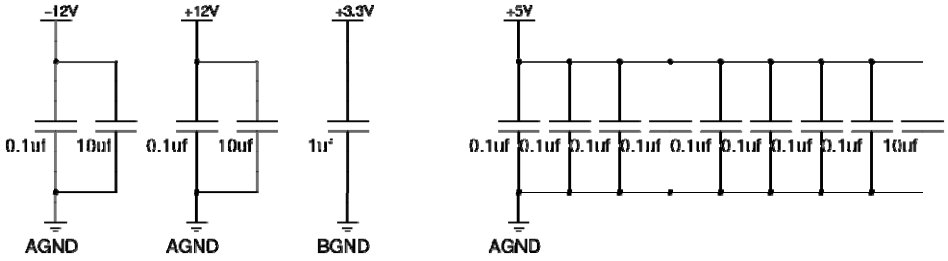


Figura 4: Condensadores del conversor

Los condensadores (Figura 4) necesarios para acondicionar el conversor se obtienen de los datos proporcionados por el fabricante del conversor en su datasheet.

2.2.3 CONEXIÓN CON LA FPGA

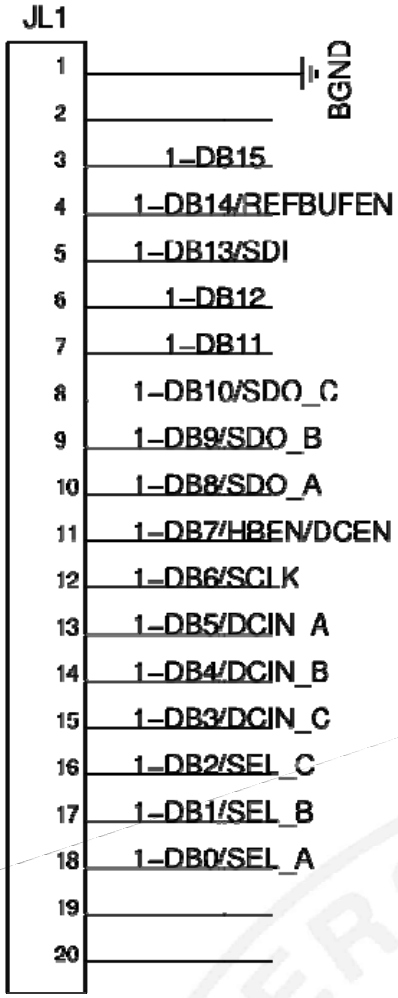


Figura 5: Conexión datos del conversor

El envió de los 16 bits de datos a la FPGA (Figura 5) se hace a través de un conector para cada conversor, el banco de conexiones se utiliza únicamente para el envió de datos.

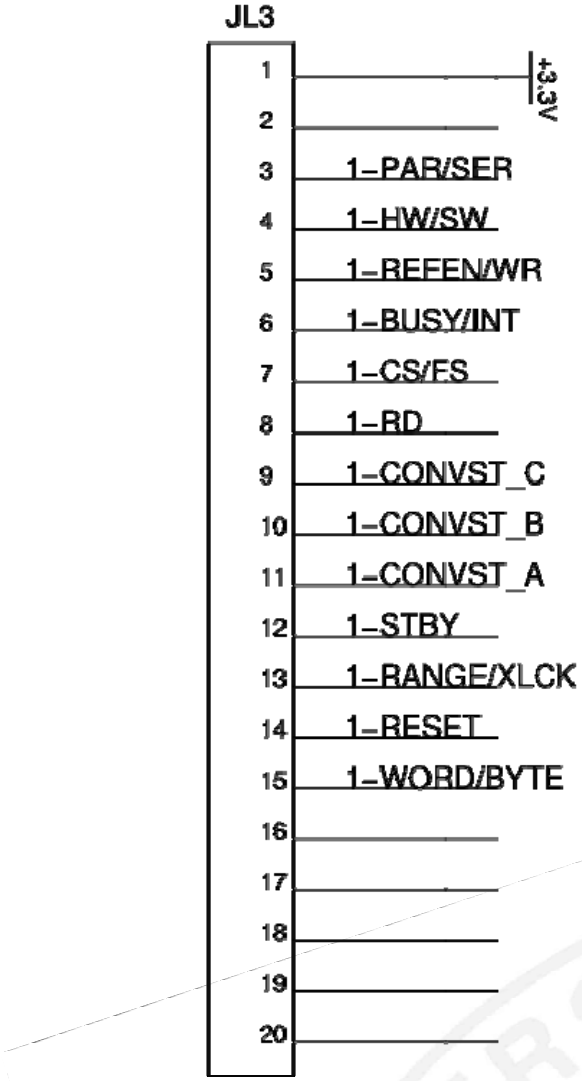


Figura 6: Conexión configuración del conversor

Todas las patillas del conversor dedicadas a la configuración del conversor que puede tomar diferentes valores han sido conectadas a la FPGA para poder configurar al gusto desde la FPGA que es más sencillo de modificar si a posteriori queremos modificar la configuración de los conversores. Se utiliza también un conector para cada conversor.

2.2.4 CONEXIONES DEL CONVERSOR

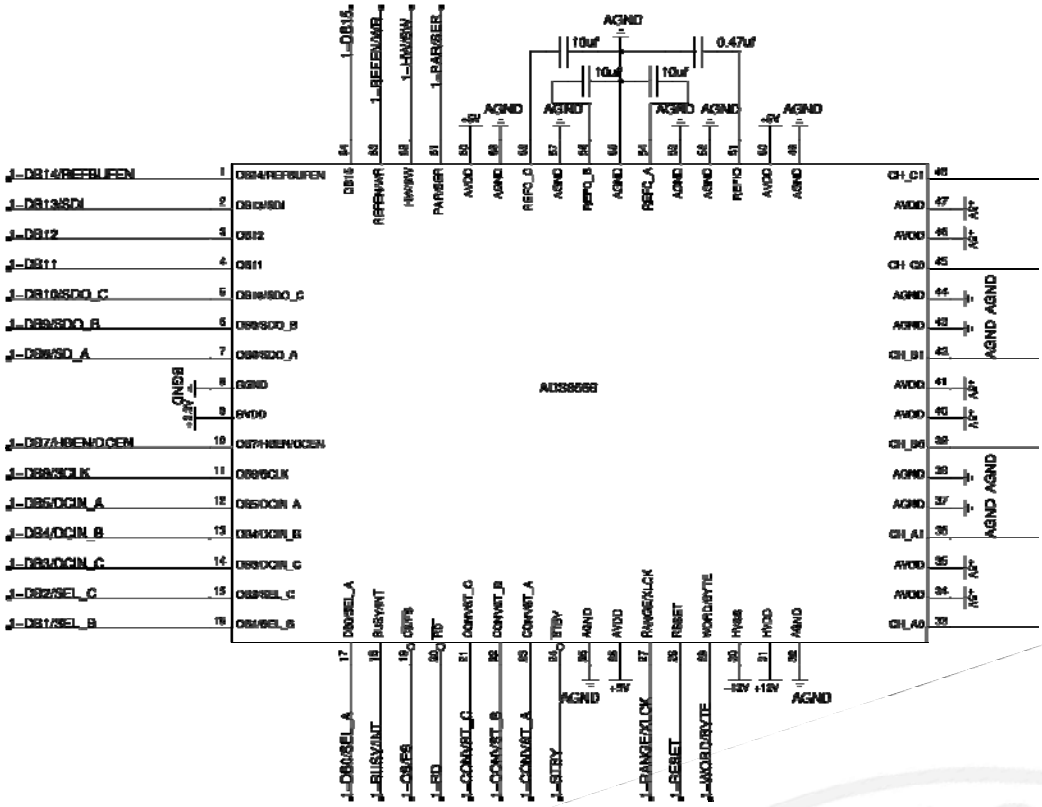


Figura 7: Conexiones del conversor

La mayoría de las patillas del conversor (Figura 7) están conectadas a los bancos de conexiones de la FPGA anteriormente comentados, seis de ellas son las seis entradas de las señales analógicas a muestrear, y el resto de las patillas están conectadas a tierra, a señales de referencia o condensadores para acondicionar el conversor según lo indica el datasheet correspondiente.

2.3 DISEÑO DEL PCB

Para el diseño del PCB se ha decidido utilizar un sistema de cuatro capas distintas, una de ellas utilizada como capa de tensión de cinco voltios, otra como capa de tierra, otra utilizada para conectar los elementos y la mayoría de las pistas y la ultima para poner otras pistas para permitir que se puedan cruzar las pistas. Se utilizarán las indicaciones de diseño Eurocircuits para que se pueda mandar fabricar a esta misma empresa, utilizando las Design Guidelines de Enero de 2010.

Pattern class	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Service	P+S+T+I+O	P+S+T+I+O	P+S+T+I+O	P+S+T+I+O	T+O	T+O	O
OTW	0.300	0.250	0.200	0.150	0.125	0.100	0.090
OTT-OTP-OPP	0.300	0.250	0.200	0.150	0.125	0.100	0.090
OAR	0.200	0.150	0.150	0.125	0.125	0.100	0.100
ITW	0.300	0.250	0.200	0.150	0.125	0.100	0.090
ITT-ITP-IPP	0.300	0.250	0.200	0.150	0.125	0.100	0.090
IAR	0.200	0.200	0.200	0.175	0.150	0.150	0.125

Tabla 5: Diseño de PCBs

A la vista de las opciones disponibles (Tabla 5) se ha elegido el pattern class 8, mínimo necesario para poder realizar el PCB utilizando los conversores analógicos digitales que hemos seleccionado. La Tabla 5 muestras las distancias permitidas entre los diferentes elementos en el circuito (Figura 8) la primera letra que es una I ó una O que significa si se refiere a una capa exterior (Outer) o a una capa interior (Inner), el campo service hace referencia a la disponibilidad del tipo de proceso con los siguientes significados:

P→ PCB proto.

S→ Estandard pool

T→ Tech pool

I→ IMS pool

O→ On demand

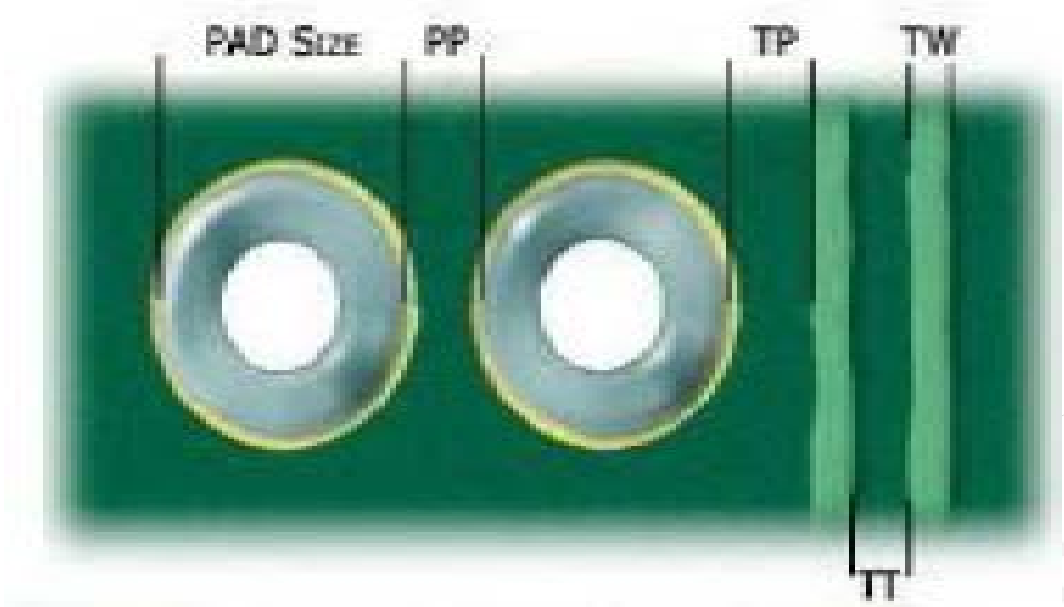


Figura 8: Distancias en el PCB

2.4 DISEÑO DE LA FPGA

El diseño de la FPGA se puede dividir en dos fases fácilmente diferenciables, por un lado la comunicación entre la FPGA y los dos conversores y por el otro la comunicación, mediante el puerto PCI, entre la FPGA y el ordenador.

El diagrama de bloques de nuestro sistema seria el que sigue:

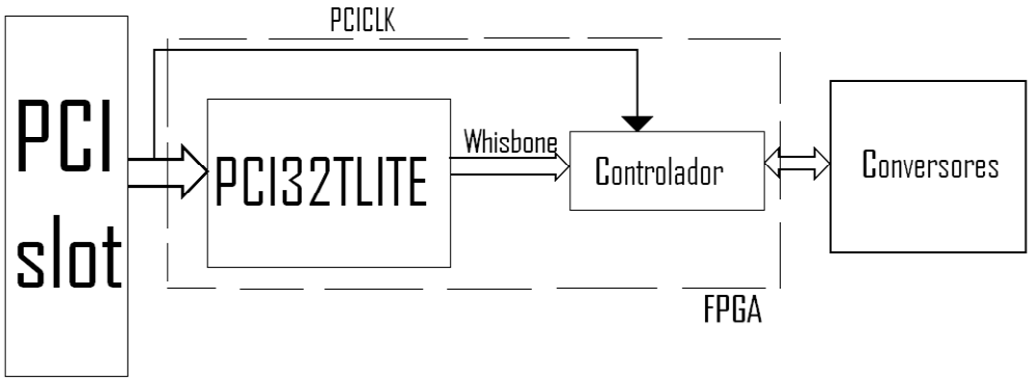


Figura 9: Diagrama de bloques

PCI Slot.

Esto es simplemente la conexión física en el ordenador del puerto PCI

PCI32TLITE.

Esto es el core que se descargará en la FPGA que se encarga de hacer de driver del puerto PCI. Este core es proporcionado por Opencores y es un core gratuito.

Bus Wishbone.

Se ha decidido utilizar el bus Wishbone para la comunicación entre el core del PCI y la FPGA, se ha tomado esta decisión, porque este bus dispone de todas las funciones necesarias para la ejecución de las especificaciones del proyecto y además ya existe su código VHDL que está incluido en el driver de la FPGA.

Controlador.

Esta es la parte principalmente diseñada de forma específica en este proyecto, se encarga de recibir las órdenes del ordenador y trasmitirlas a los conversores, también se encarga de recibir la información de los conversores y devolverla al ordenador cuando corresponda de forma ordenada, el sistema contará con un banco de registros donde almacenará los resultados de las conversiones así como las opciones de configuración de los conversores. Una maquina de estados se encargará de controlar el correcto funcionamiento de los conversores.

Conversores.

Estos son los dos conversores analógico digital ADS8556 de Texas Instruments.

2.4.1 PCI32TLITE

Esto es el core que se descargará en la FPGA que se encarga de hacer de driver del puerto PCI, este core es proporcionado por Opencores, se utilizará el código de libre acceso proporcionado por Varxec en su versión tres, haciendo aquellas modificaciones que se consideren necesarias.

En primer lugar se tiene que configurar el bus PCI para que el ordenador reconozca el dispositivo y viceversa, durante esta fase se almacenan en el ordenador datos (Figura 10) como el fabricante del chip y de la tarjeta, que pueden diferir, así como el identificador de chip y tarjeta, tipo de tarjeta, direcciones base de los registros, características, comandos soportados y tamaño del cache. Hecho esto le queda asignada a la tarjeta PCI una dirección inequívoca que utilizará el ordenador para la comunicación. Este proceso se realiza automáticamente al encenderse el ordenador.

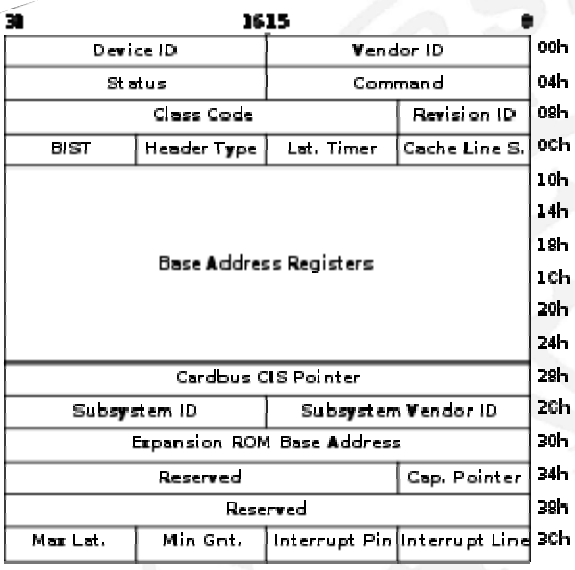


Figura 10: Configuración inicial

De las posibles funciones del puerto PCI utilizaremos únicamente las más sencillas, el ciclo de lectura simple (Figura 11) y el ciclo de escritura simple (Figura 12).

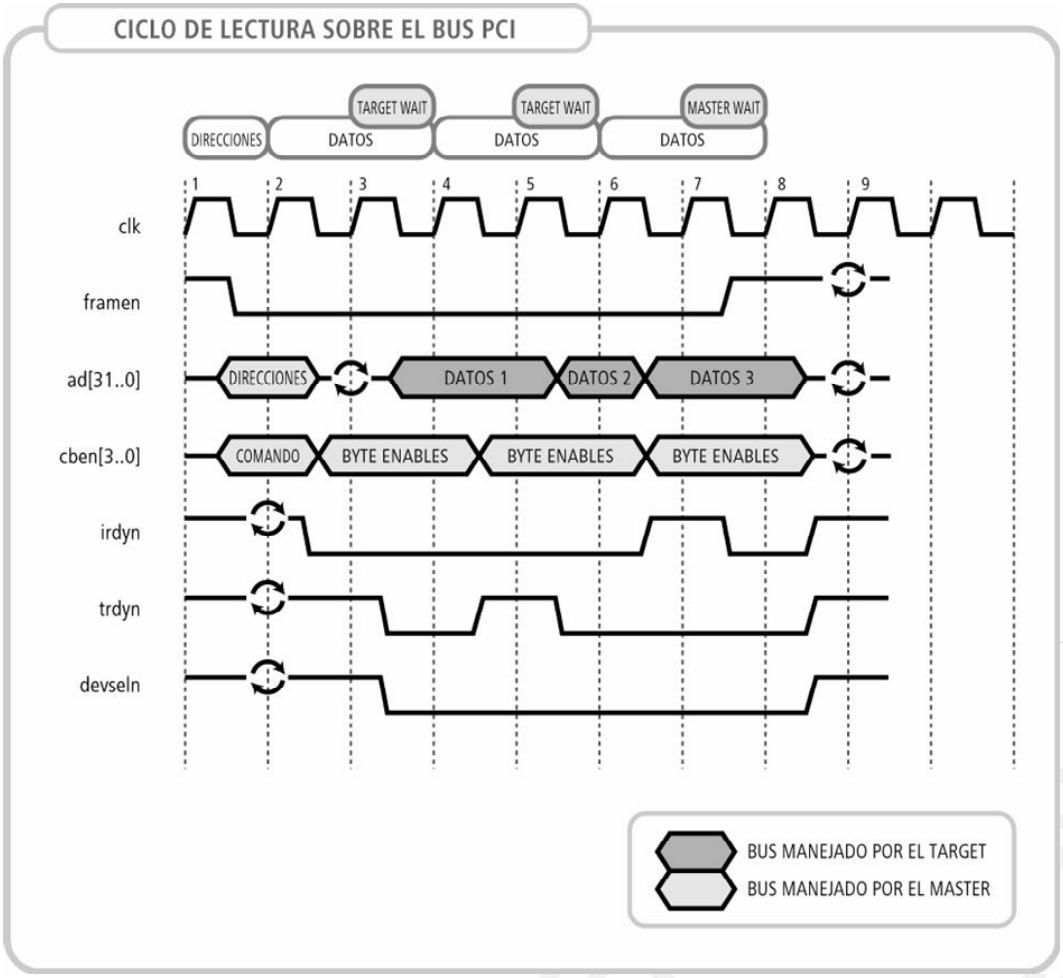


Figura 11: Ciclo de lectura simple bus PCI

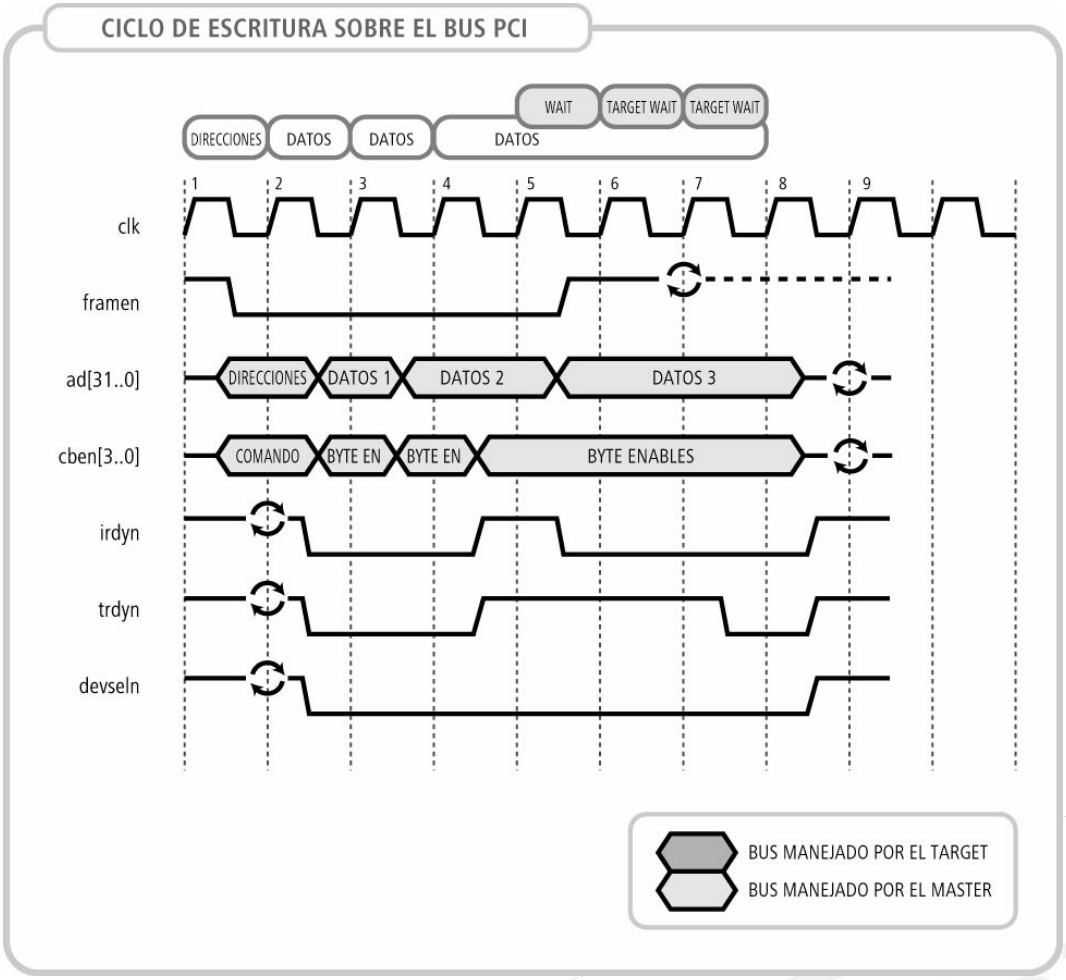


Figura 12: Ciclo de escritura simple bus PCI

2.4.2 BUS WISHBONE

2.4.2.1 Introducción

El bus wishbone es una metodología flexible diseñada para usar con cores de semiconductores. Se trata de crear una interfaz común entre cores que aporte portabilidad al sistema y resulte en una solución rápida para el usuario final. Antes de la estandarización de este bus la interconexión era complicada y requería de mucho tiempo para diseñar la unión entre los dos sistemas, con la estandarización, es fácil y sencillo conectar cualquier sistema con un bus wishbone a otro.

Algunas características importantes del bus son:

_Simple y compacto, requiere muy pocas puertas lógicas.

_Dispone de todos los tipos de transferencia de datos habituales.

_Bus de datos regulable hasta los 64 bits.

_Soporta organización de datos Big Endian y Little Endian.

_Vectores de interrupción.

_Arquitectura maestro-esclavo.

_Configuraciones multimaestro.

Aunque el bus dispone de múltiples funciones y de múltiples modos de funcionamiento, para el desarrollo de este proyecto será suficiente con el método más sencillo. Usaremos el ciclo de lectura y escritura simples, por lo que no utilizaremos muchas de las señales proporcionadas por el bus al no ser necesarias para este cometido.

Se ha decidido utilizar el bus Wishbone para la comunicación entre el puerto PCI y la FPGA, se ha tomado esta decisión, porque este bus dispone de todas las funciones necesarias para la ejecución de las especificaciones del proyecto y además ya existe su código VHDL que está incluido en el driver de la FPGA.

2.4.2.2 Señales utilizadas

El bus Wishbone dispone de muchas más funciones de la que necesitamos para realizar este proyecto, por lo que no utilizaremos todas las entradas y salidas disponibles en el mismo (Figura 13). Utilizaremos las siguientes entradas:

_RST_I: esta entrada permite inicializar el estado del bus en un estado conocido

_CLK_I: Entrada de reloj para controlar los tiempos del bus

_DAT_I0: Por esta entrada entrarán aquellos datos que queramos que sean mandados al ordenador por medio del puerto PCI

Utilizaremos las siguientes salidas:

_ADR_O0: Por esta salida enviaremos la dirección de los registros del sistema al que estamos haciendo referencia en la orden que estemos ejecutando desde el ordenador en ese momento.

_DAT_O0: Por esta salida enviaremos los datos de configuración de los conversores a si como los referentes al tipo de conexión y la inicialización o el paro de la conversión y cualquier otro tipo de información necesaria.

_WE_O: Esta salida nos indicará si nos encontramos ante un ciclo de lectura o de escritura en el bus, si nos encontramos ante un ciclo de lectura nos encontraremos un cero lógico, mientras que en los ciclos de escritura nos encontraremos con un uno lógico.

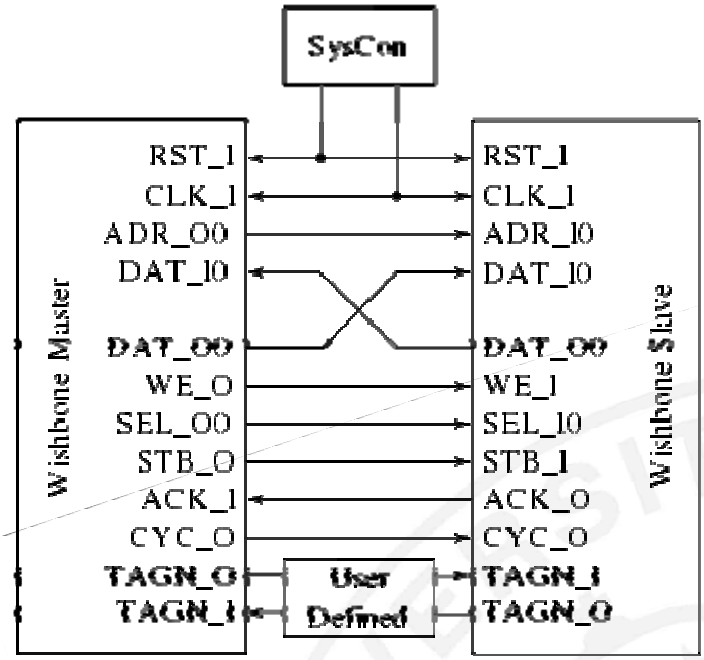


Figura 13: Bus wishbone

2.4.2.3 Operaciones generales.

2.4.2.3.1 Ciclo de lectura (Figura 14)

Clock edge 0:

El maestro pone una dirección valida en [ADR_O()] y [TGA_O()].

El maestro niega [WE_O] para indicar que es un ciclo de lectura.

El maestro selecciona el banco [SEL_O()] para indicar dónde está el dato.

El maestro pone a uno [CYC_O] y [TGC_O()] para indicar el inicio del ciclo.

El maestro pone a uno [STB_O] para indicar el inicio de la fase.

Clock edge 1:

El esclavo decodifica la entrada y responde poniendo a uno [ACK_I].

El esclavo pone datos validos en [DAT_I()] y [TGD_I()].

El esclavo pone a uno [ACK_I] en repuesta a [STB_O] para indicar que el dato es válido.

El maestro monotoriza [ACK_I] y se prepara para coger los datos en [DAT_I()] y [TGD_I()].

Clock edge 2:

El maestro coge los datos en [DAT_I()] y [TGD_I()].

El maestro niega [STB_O] y [CYC_O] para indicar el final del ciclo.

El esclavo niega [ACK_I] en repuesta a [STB_O].

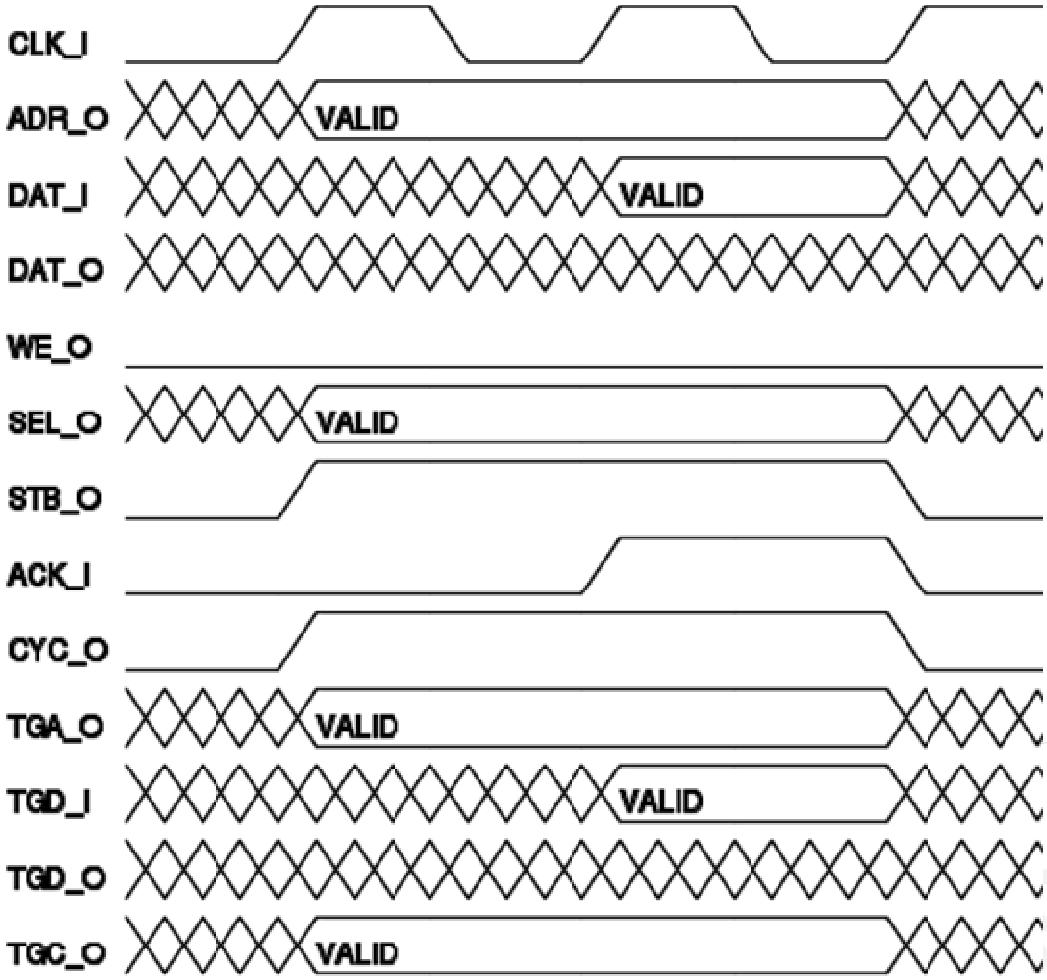


Figura 14: Ciclo de lectura

2.4.2.3.2 Ciclo de escritura (Figura 15)

Clock edge 0:

El maestro pone una dirección válida en [ADR_O()] y [TGA_O()].

El maestro pone un dato válido en [DAT_O()] y [TGD_O()].

El maestro pone a uno [WE_O] para indicar que es un ciclo de escritura.

El maestro selecciona el banco [SEL_O()] para indicar dónde manda el dato.

El maestro pone a uno [CYC_O] y [TGC_O()] para indicar el inicio del ciclo.

El maestro pone a uno [STB_O] para indicar el inicio de la fase.

Clock edge 1:

El esclavo decodifica la entrada y responde poniendo a uno [ACK_I].

El esclavo se prepara para coger datos en [DAT_O()] y [TGD_O()].

El esclavo pone a uno [ACK_I] en repuesta a [STB_O] para indicar que está preparado.

El maestro monitoriza [ACK_I] y se prepara para terminar el ciclo.

Clock edge 2:

El esclavo coge los datos en [DAT_O()] y [TGD_O()].

El maestro niega [STB_O] y [CYC_O] para indicar el final del ciclo.

El esclavo niega [ACK_I] en repuesta a [STB_O].

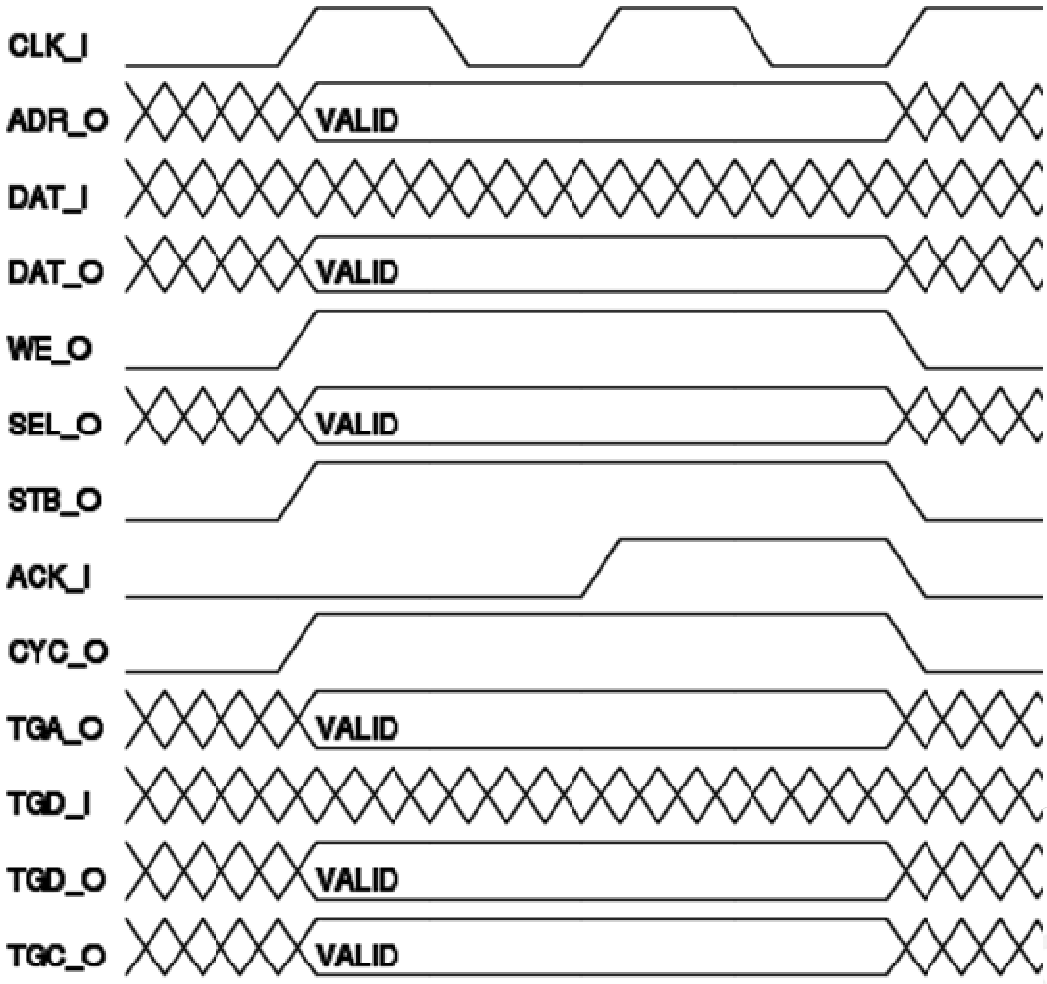


Figura 15: Ciclo de escritura

2.4.3 CONTROLADOR

2.4.3.1 Introducción

Esta es la parte principalmente de la FPGA diseñada de forma específica en este proyecto, se encarga de recibir las órdenes del ordenador y trasmitirlas a los conversores, también se encarga de recibir la información de los conversores y devolverla al ordenador cuando corresponda de forma ordenada. El sistema contará con un banco de registros donde almacenará los resultados de las conversiones así como las opciones de configuración de los conversores (Figura

16). Varias maquinas de estados se encargan de controlar el correcto funcionamiento del circuito.

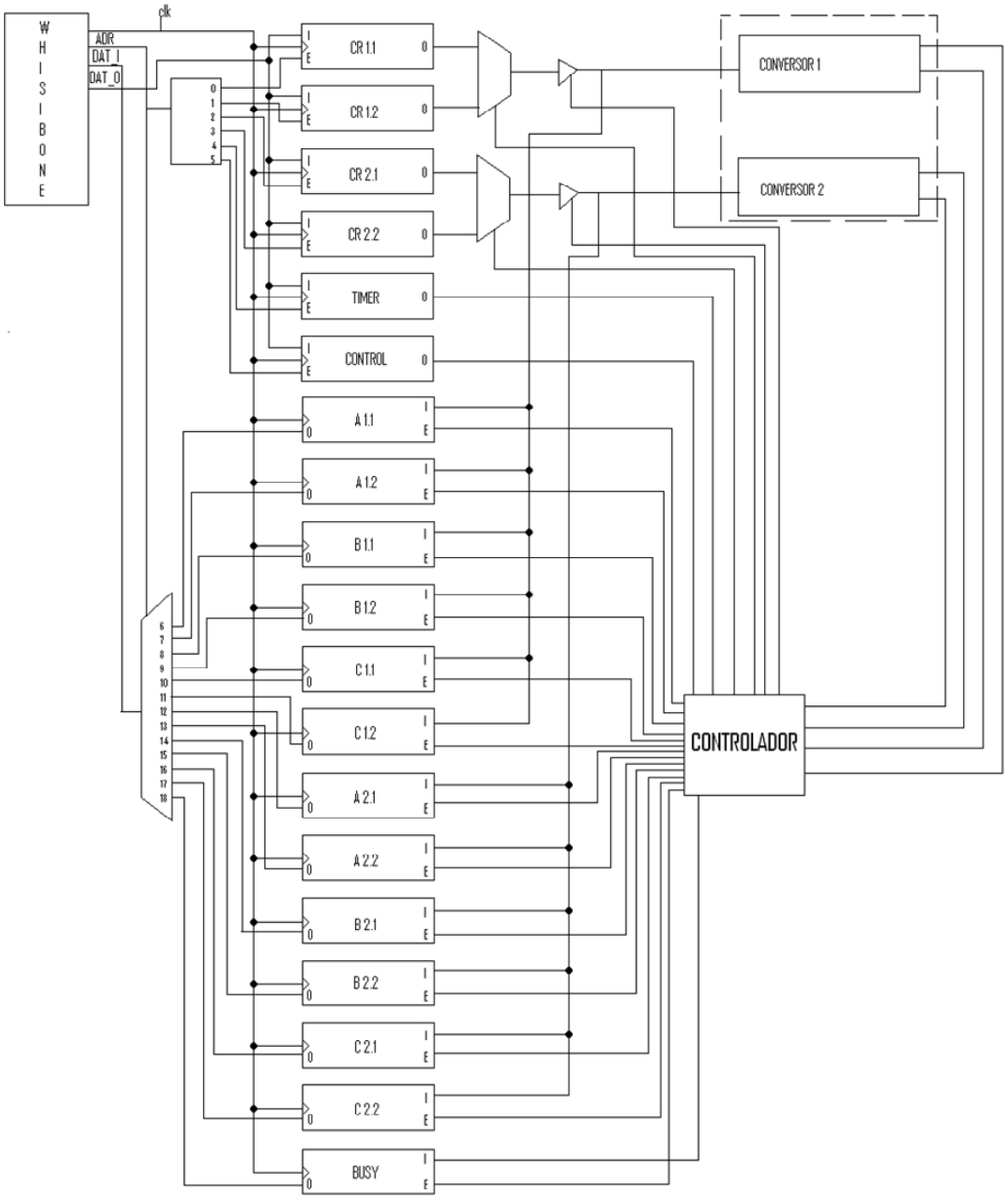


Figura 16: Esquema lógico

2.4.3.2 Explicación de los registros

2.4.3.2.1 Registros tipo CR X.Y

Estos registros son los encargados de almacenar la información sobre la configuración de los conversores, guardan la disposición del conversor X(1 o 2) su parte Y, siendo 1 los bits menos significativos (Tabla 6) y 2 los más significativos (Tabla 7).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PD_C	PD_B	PD_A	0	CLKSEL	CLKOUT_EN	REFDAC(9-0)									

Tabla 6: Registro CR X.1

15	14	13	12	11	10	9	8
CH_C	CH_B	CH_A	RANGE_C	RANGE_B	RANGE_A	REF_{EN}	REFBUF
7	6	5	4	3	2	1	0
SEQ	A-NAP	BUSY/INT	BUSY L/H	0	VREF	READ_EN	C23:0_EN

Tabla 7: Registro CR X.2

Bit	*Nombre*	*Descripción*
C31	CH_C	0 = Channel pair C disabled for next conversion (default) 1 = Channel pair C enabled
C30	CH_B	0 = Channel pair B disabled for next conversion (default) 1 = Channel pair B enabled
C29	CH_A	0 = Channel pair A disabled for next conversion (default) 1 = Channel pair A enabled
C28	RANGE_C	0 = Input voltage range selection for channel pair C: 4VREF (default) 1 = Input voltage range selection for channel pair C: 2VREF
C27	RANGE_B	0 = Input voltage range selection for channel pair B: 4VREF (default) 1 = Input voltage range selection for channel pair B: 2VREF
C26	RANGE_A	0 = Input voltage range selection for channel pair A: 4VREF (default) 1 = Input voltage range selection for channel pair A: 2VREF
C25	REF_{EN}	0 = Internal reference source disabled (default) 1 = Internal reference source enabled
C24	REFBUF	0 = Internal reference buffers enabled (default) 1 = Internal reference buffers disabled
C23	SEQ	0 = Sequential convert start mode disabled (default) 1 = Sequential convert start mode enabled (bit 11 must be '1' in this case)

C22	A-NAP	0 = Normal operation (default) 1 = Auto-NAP feature enabled
C21	BUSY/INT	0 = BUSY/INT pin in normal mode (BUSY) (default) 1 = BUSY/INT pin in interrupt mode (INT)
C20	BUSY L/H	0 = BUSY active high while INT active low (default) 1 = BUSY active low while INT active high
C19	Sin uso	This bit is always set to '0'
C18	VREF	0 = Internal reference voltage: 2.5V (default) 1 = Internal reference voltage: 3V
C17	READ_EN	0 = Normal operation (conversion results available on SDO_x) (default) 1 = Control register contents output on SDO_x with next access
C16	C23:0_EN	0 = Control register bits C[31:24] update only (serial mode only) (default) 1 = Entire control register update enabled (serial mode only)
C15	PD_C	0 = Normal operation (default) 1 = Power-down for channel pair *C* enabled (bit 31 must be '0' in this case)
C14	PD_B	0 = Normal operation (default) 1 = Power-down for channel pair *B* enabled (bit 31 must be '0' in this case)
C13	PD_A	0 = Normal operation (default) 1 = Power-down for channel pair A enabled (bit 31 must be '0' in this case)
C12	Sin uso	This bit is always '0'
C11	CLKSEL	0 = Normal operation with internal conversion clock (default) 1 = External conversion clock (applied through pin 27) used
C10	CLKOUT_EN	0 = Normal operation (default) 1 = Internal conversion clock available at pin 27
C09	REFDAC(9)	Bit 9 (MSB) of reference DAC value; default = 1
C08	REFDAC(8)	Bit 8 of reference DAC value; default = 1
C07	REFDAC(7)	Bit 7 of reference DAC value; default = 1
C06	REFDAC(6)	Bit 6 of reference DAC value; default = 1
C05	REFDAC(5)	Bit 5 of reference DAC value; default = 1
C04	REFDAC(4)	Bit 4 of reference DAC value; default = 1

C03	REFDAC(3)	Bit 3 of reference DAC value; default = 1
C02	REFDAC(2)	Bit 2 of reference DAC value; default = 1
C01	REFDAC(1)	Bit 1 of reference DAC value; default = 1
C00	REFDAC(0)	Bit 0 of reference DAC value; default = 1

2.4.3.2.2 Registros tipo ZXY

Estos registros almacenan los datos del canal Y(1 o 2) del par de canales Z(A, B o C) del conversor X(1 o 2) (Tabla 8). Es decir el registro B21 será el que almacene la información del canal 1 del par de canales B del conversor 2.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATO(15-0)															

Tabla 8: Registro XYZ

2.4.3.2.3 Registro TIMER

Este registro guarda la información sobre el tiempo que se deben de ejecutar conversiones (Tabla 9).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TIMER(15-0)															

Tabla 9: Registro TIMER

2.4.3.2.4 Registro CONTROL

Este registro (Tabla 10) guarda información sobre el tipo de conversión deseada así como directrices para empezar o terminar la conversión.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	PARADA	CONTINUA	UNICA

Tabla 10: Registro CONTROL

Unica: Si este bit esta a 1 se realiza una única conversión.

Continua: Si este bit esta a 1 se realiza una conversión continúa.

Parada: Si este bit esta a 1 indica que se debe detener la conversión.

2.4.3.2.5 Registro BUSY

Este registro (Tabla 11) informa al ordenador de si la conversión ha finalizado.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUSY	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 11: Registro BUSY

2.4.3.3 Máquinas de estado.

2.4.3.3.1 Máquina de estados principal.

Esta máquina de estados (Figura 17) se encarga de controlar el resto de las maquinas de estado presentes en el sistema.

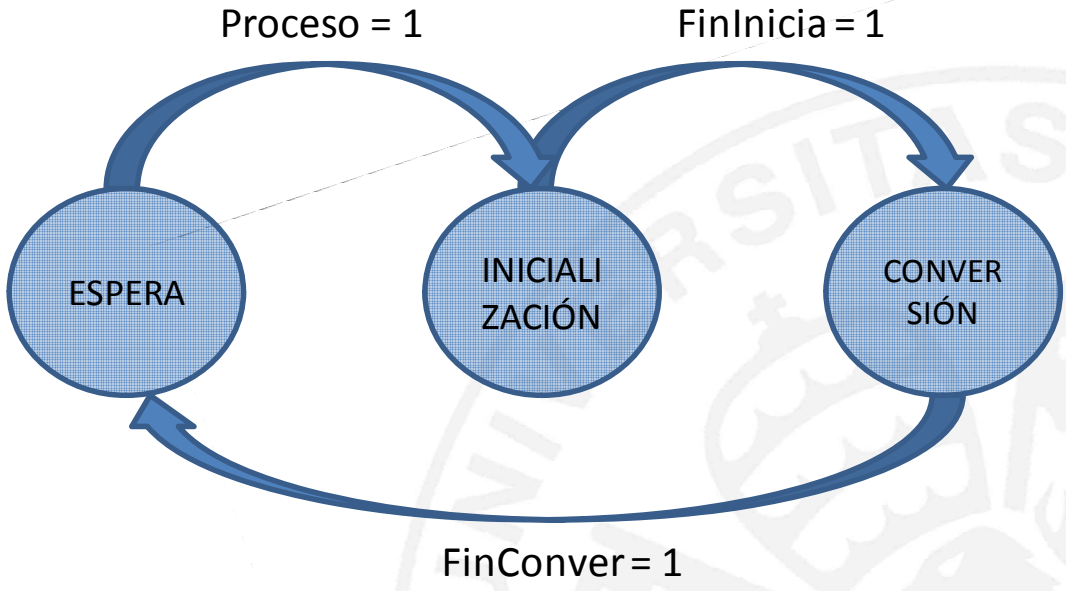


Figura 17: Maquina de estados principal

_Estado Espera: El sistema está a la espera de recibir la orden de realizar una o más conversiones

_Estado Inicialización: El sistema está a la espera de recibir la señal de que la inicialización de los conversores ha terminado

_Estado Conversión: El sistema está a la espera de recibir la señal de que la conversión o conversiones han terminado.

_Transición Espera-Inicialización: Al recibir la señal de inicio de conversión, Proceso=1, pone Inicia=1

_Transición Inicialización-Conversión: Al recibir la señal de fin de inicialización, FinInicia=1, pone Conver=1

_Transición Conversión-Espera: Al recibir la señal de fin de conversión, FinConver=1, pone Proceso=0

2.4.3.3.2 Máquina de estados de la inicialización.

Esta máquina de estados (Figura 18) se encarga de configurar los registros de los conversores para realizar las conversiones de acuerdo a nuestras condiciones.

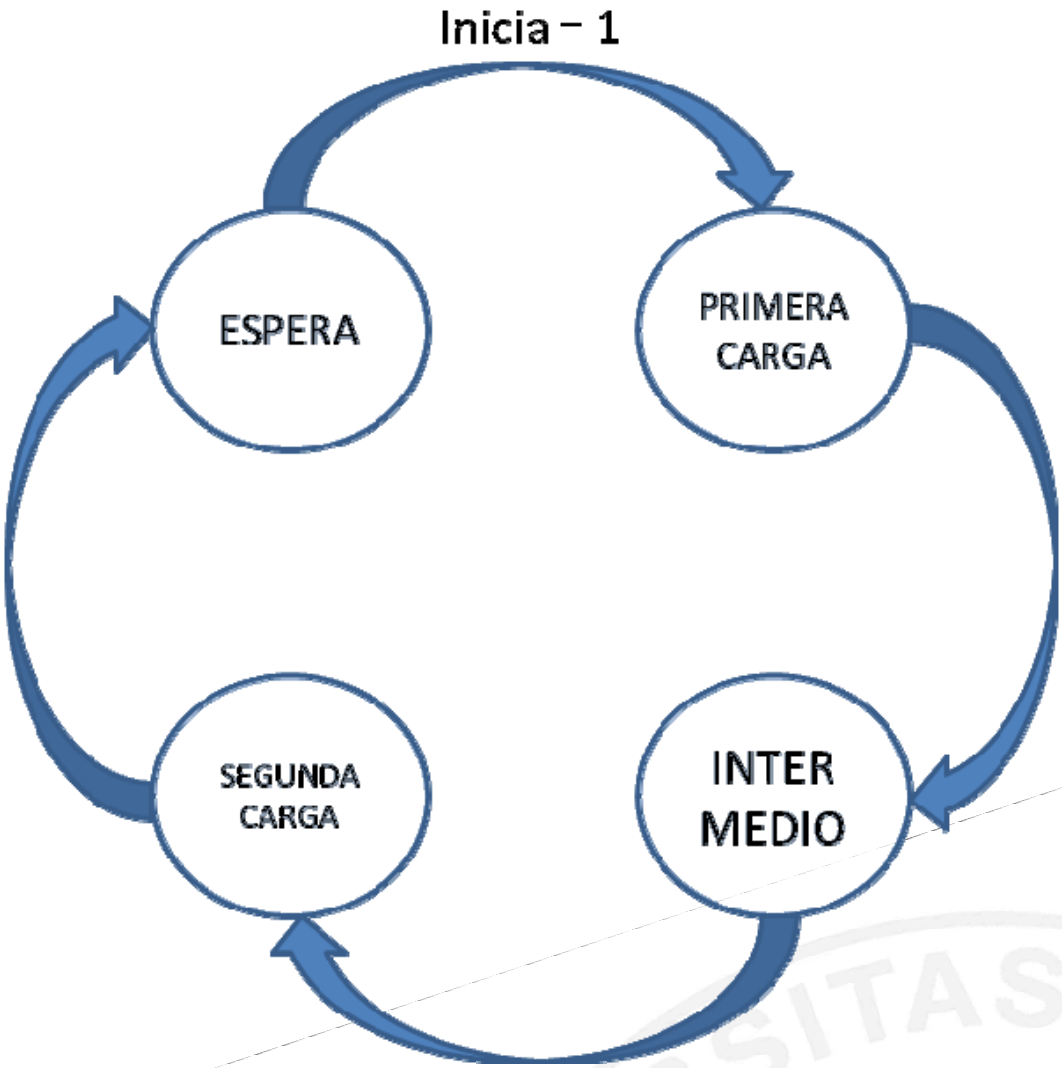


Figura 18: Maquina de estados de la inicialización

_Estado Espera: El sistema está a la espera de configurar los conversores

_Estado Primera carga: El sistema carga la parte alta del registro de control de los dos conversores

_Estado Intermedio: El sistema espera el tiempo necesario para el correcto funcionamiento del conversor.

_Estado Segunda carga: El sistema carga la parte baja del registro de control de los dos conversores

_Transición Espera-Primera carga: Al recibir la señal de inicio de inicialización, Inicia=1, pone CS=1, WR=1, CR1=1, CR2=1, Escritura=1.

_Transición Primera carga-Intermedio: pone WR=0, InicioT2=1

_Transición Intermedio-Segunda carga: pone WR=1, InicioT1=1

_Transición Segunda carga-Espera: pone CS=0, WR=0, Inicia=0

2.4.3.3.3 Maquina de estados de la conversión.

Esta máquina de estados (Figura 19) se encarga de administrar todas las señales necesarias para realizar la conversión así como de almacenar los datos obtenidos en los registros correspondientes.

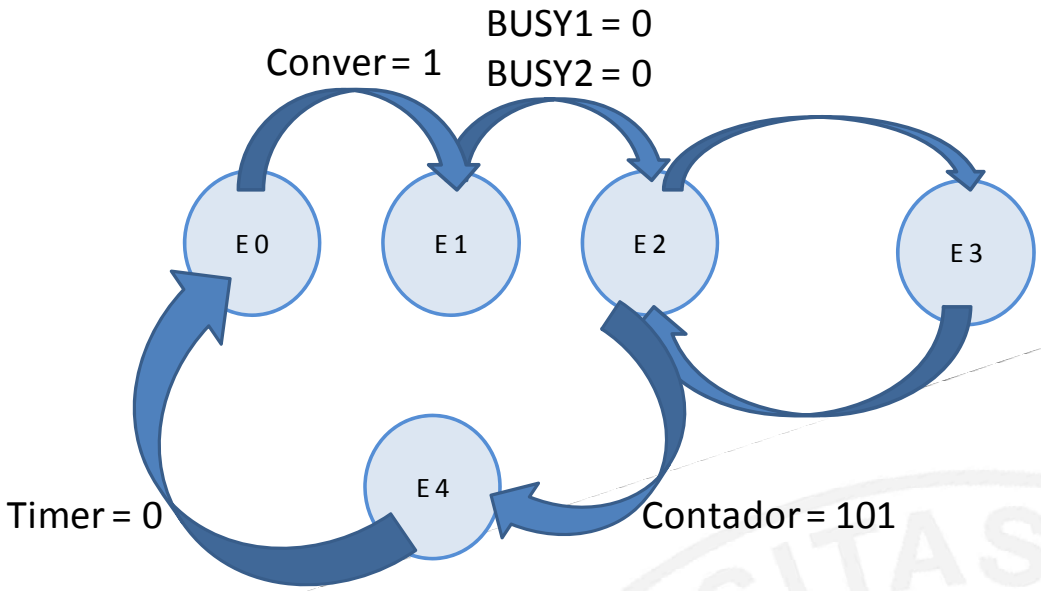


Figura 19: Maquina de estados de la conversión

_Estado cero: Estado de espera, el sistema está a la espera de una nueva orden para realizar una conversión.

_Estado uno: La FPGA ha dado la orden a los conversores para realizar la lectura y esta a la espera de que los conversores comuniquen que han acabado la conversión.

_Estado dos: En este estado la FPGA guarda la información de dos canales, uno de cada conversor, el sistema pasa seis veces por este estado en un mismo ciclo de conversión para leer los doce canales.

_Estado tres: Estado de espera necesario por el conversor entre lectura y lectura.

_Estado cuatro: Estado de espera necesario por el conversor para poder empezar una nueva conversión.

El sistema se mueve entre los diferentes estados con las siguientes transiciones.

_Transición E0-E1: Cuando se recibe Conver=1 se produce la transición de E0 a E1 y se activan las tres señales de conversión CONVST_X de cada conversor, una por cada dos canales.

_Transición E1-E2: Cuando ambos conversores comunican que han terminado su conversión mediante la señal Busy se produce la transición de E1 a E2, se activa el chip select y el RD, se le pide a los conversores que muestren el primer dato y se inicializa un contador para contabilizar el número de canales leídos de cada conversor. Se inicia el Timer.

_Transición E2-E3: Aumenta en uno el contador de pares de canales leídos, deja de pedirle el dato a los conversores, se guarda el dato en el registro correspondiente y si el número de pares de canales leídos es menor de 6 cambia al estado E3. Se desactiva el RD.

_Transición E3-E2: Cambia al estado E2, pide el siguiente dato a los conversores. Se activa el RD.

_Transición E2-E4: Aumenta en uno el contador de canales leídos, deja de pedirle el dato a los conversores, se guarda el dato en el registro correspondiente y si el número de pares de canales leídos es igual a 6 cambia al estado a E4 y el chip select y RD se desactivan.

_Transición E4-E0: Cuando el TimerT0 ha finalizado su cuenta el sistema ha esperado el tiempo suficiente para hacer otra conversión y vuelve al estado de espera E0.

Capítulo 3 RESULTADOS/EXPERIMENTOS

Procedimiento de prueba del sistema VHDL.

Se testeará el circuito diseñado mediante un test bench. En el test bench primero se cargará el registro de ambos conversores (Figura 20) con la configuración por defecto, con todos los canales activados. Ambos registros se configurarán igual, la configuración será la siguiente:

PD_C	PD_B	PD_A	0	CLKSEL	CLKOUT_EN	REFDAC(9-0)
0	0	0	0	0	0	1111111111

CH_C	CH_B	CH_A	RANGE_C	RANGE_B	RANGE_A	REF_{EN}	REFBUF
1	1	1	0	0	0	0	0
SEQ	A-NAP	BUSY/INT	BUSY L/H	0	VREF	READ_EN	C23:0_EN
0	0	0	0	0	0	0	0

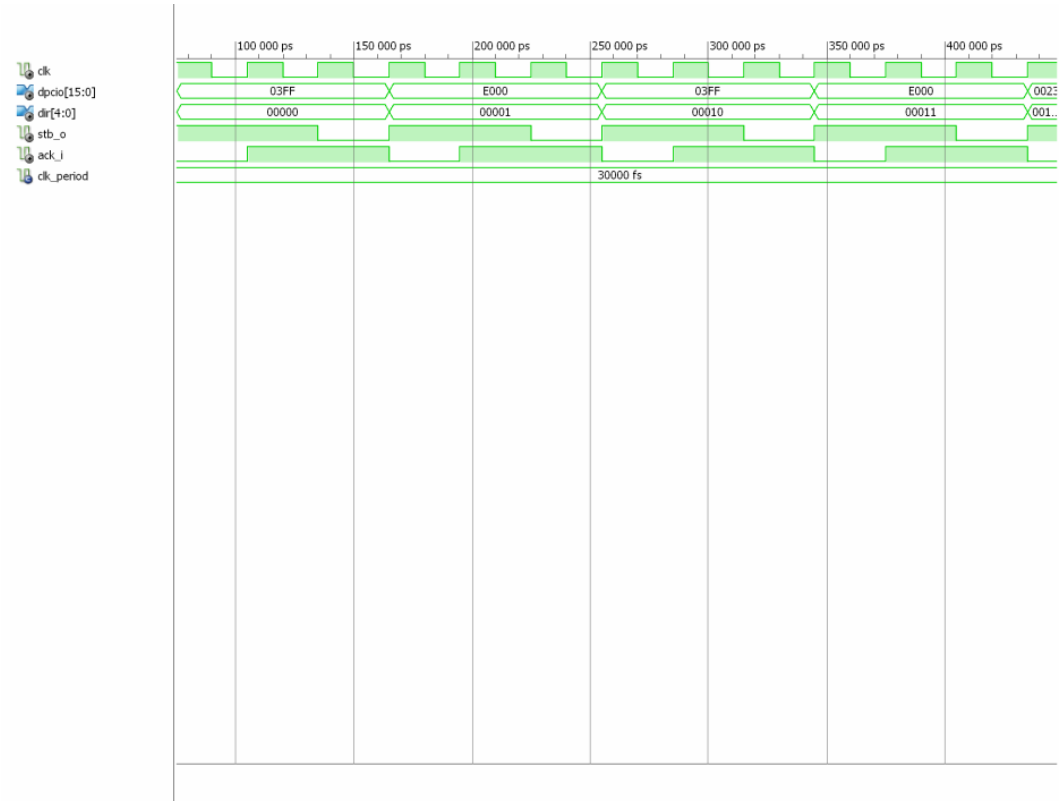


Figura 20 : Carga conversores.

Después se cargará el tiempo de ejecución para el modo continuo (Figura 21), se configura para 1080 ns que permitirá hacer dos conversiones.

TIMER(15-0)
0000000000100011

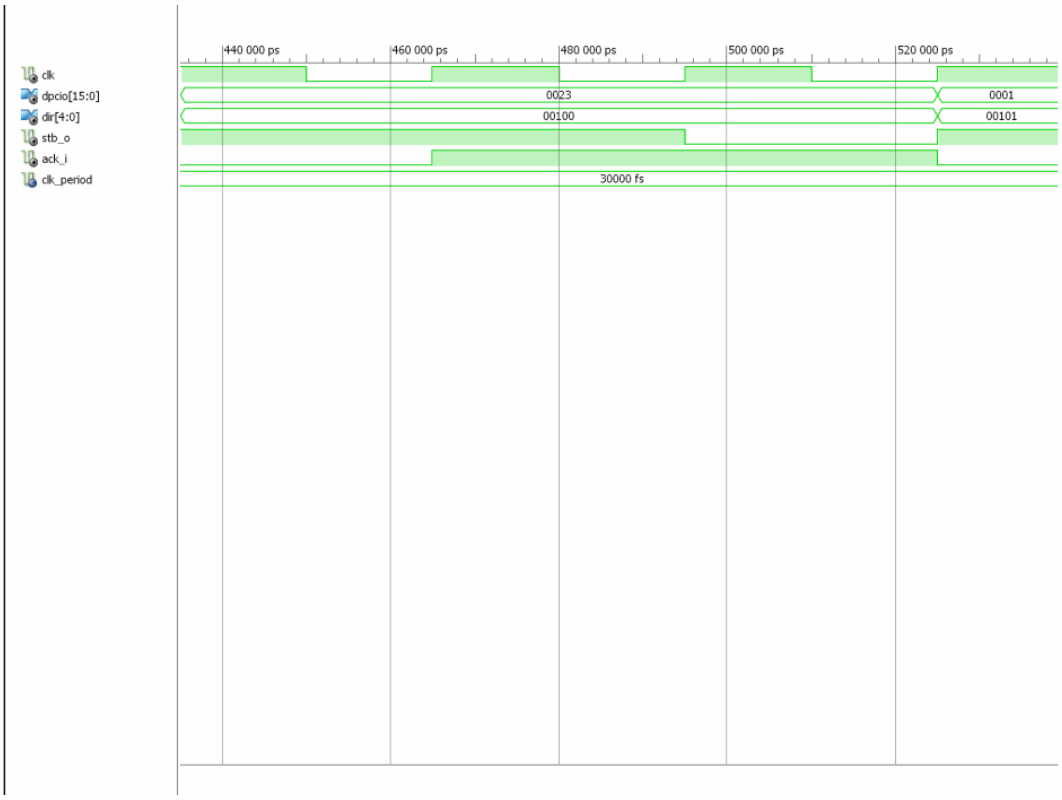


Figura 21: Carga tiempo de ejecución

Finalmente se iniciará la conversión sencilla dando la orden en el registro de control (Figura 22), y a partir de ese momento se vigilará el registro Busy para comprobar que la conversión ha finalizado.

CONTROL(15-0)
0000000000000001

Figura 22: Inicio conversión simple.

Seguidamente el Controlador cargará la información de configuración en los conversores (Figura 23).

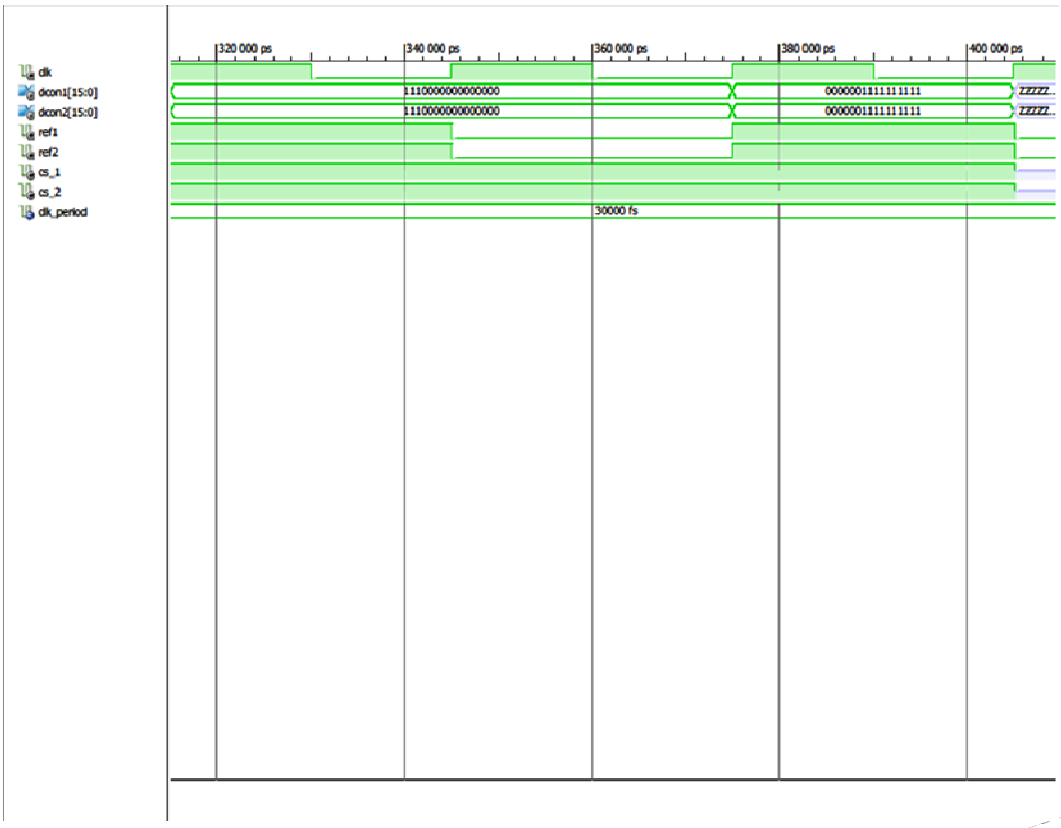


Figura 23: Configuración de los conversores

Cuando se detecte el inicio de conversión, emulando a los conversores, se iniciará la señal busy1 y busy2 y se mantendrá durante 360ns (Figura 24). Después de apagar las señales busy se esperará a que el sistema pida el primer dato y se le irán dando de forma secuencial según le sean pedidos por el sistema (Figura 25). Durante todo este tiempo se estará vigilando el registro Busy esperando a que se ponga a cero (Figura 26).

BUSY(15-0)
0000000000000000

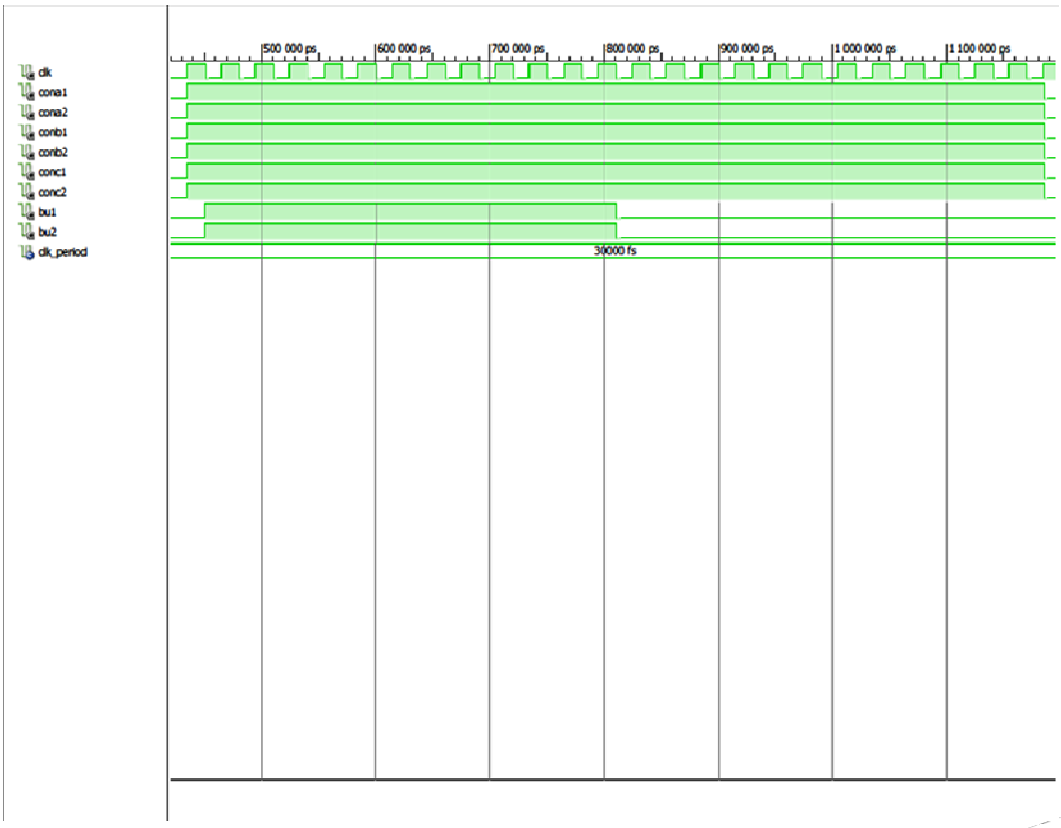


Figura 24: Tiempo de conversión.

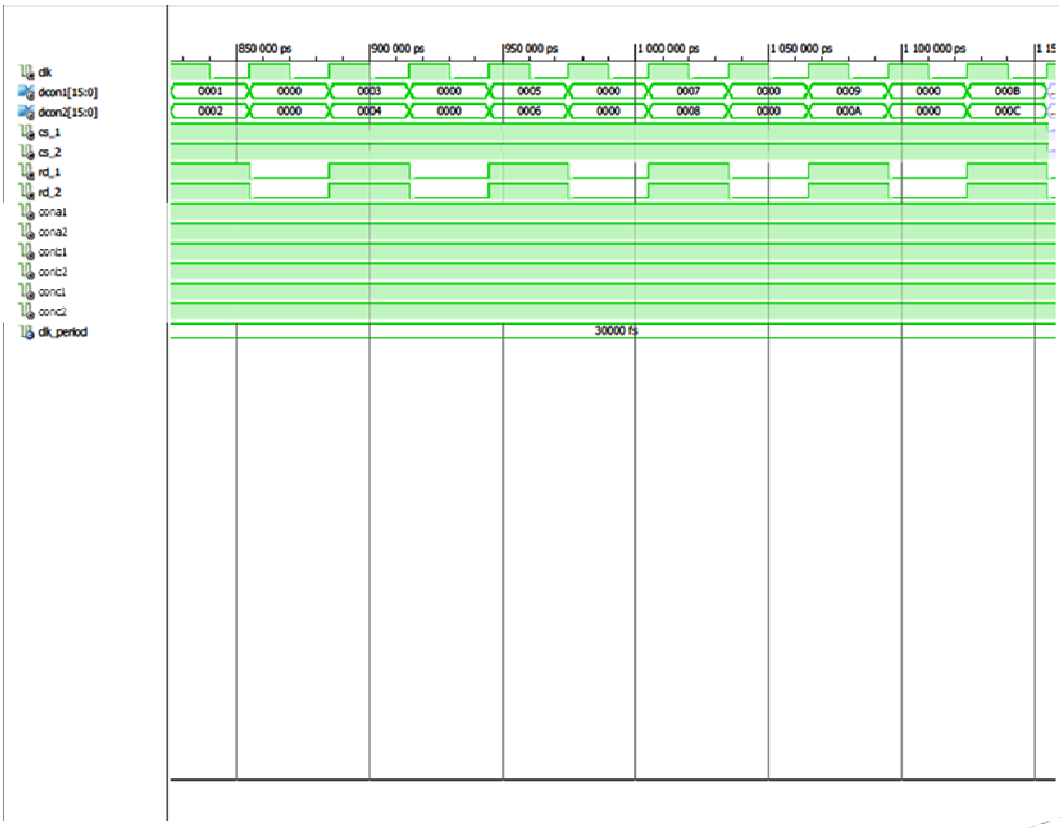


Figura 25: Lectura de los conversores

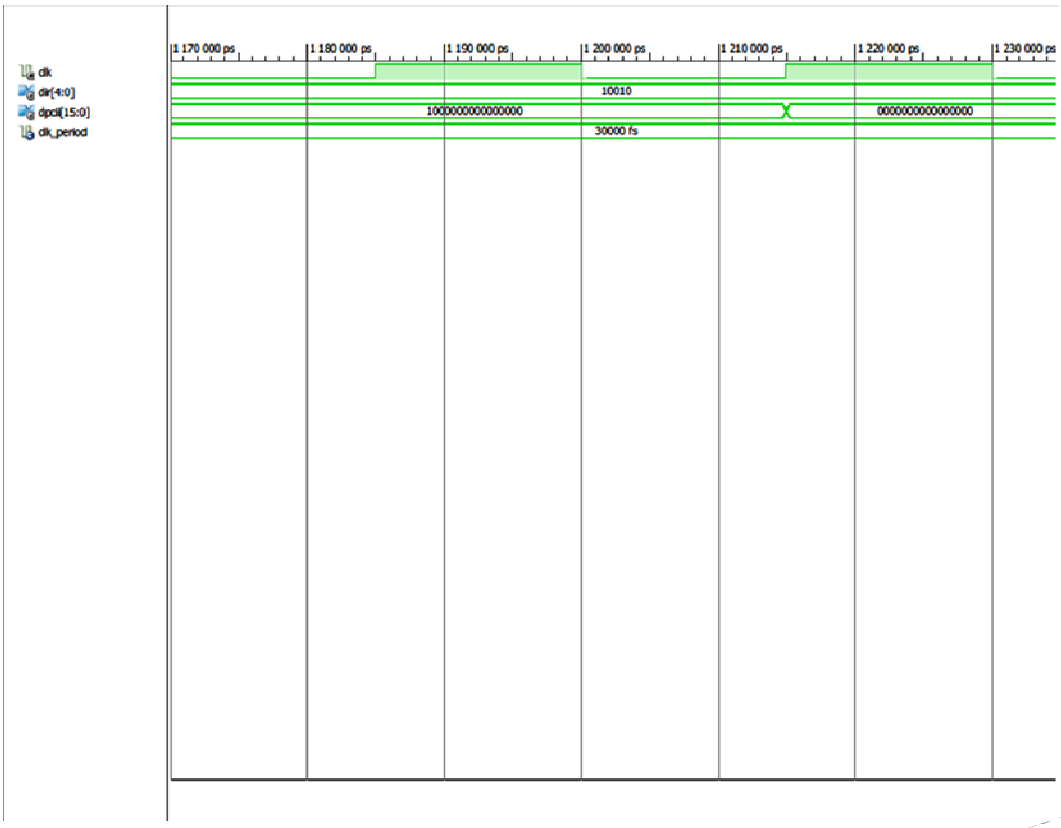


Figura 26: Fin de la conversión

Cuando esto ocurra se empezará a leer todos los registros, si los datos son correctos, los datos leídos tendrán que ir del 1 al 12 (Figura 27). Después de esto se iniciará una conversión continua que dará como resultado dos conversiones seguidas y se emulará el procedimiento antes descrito para realizar las conversiones.

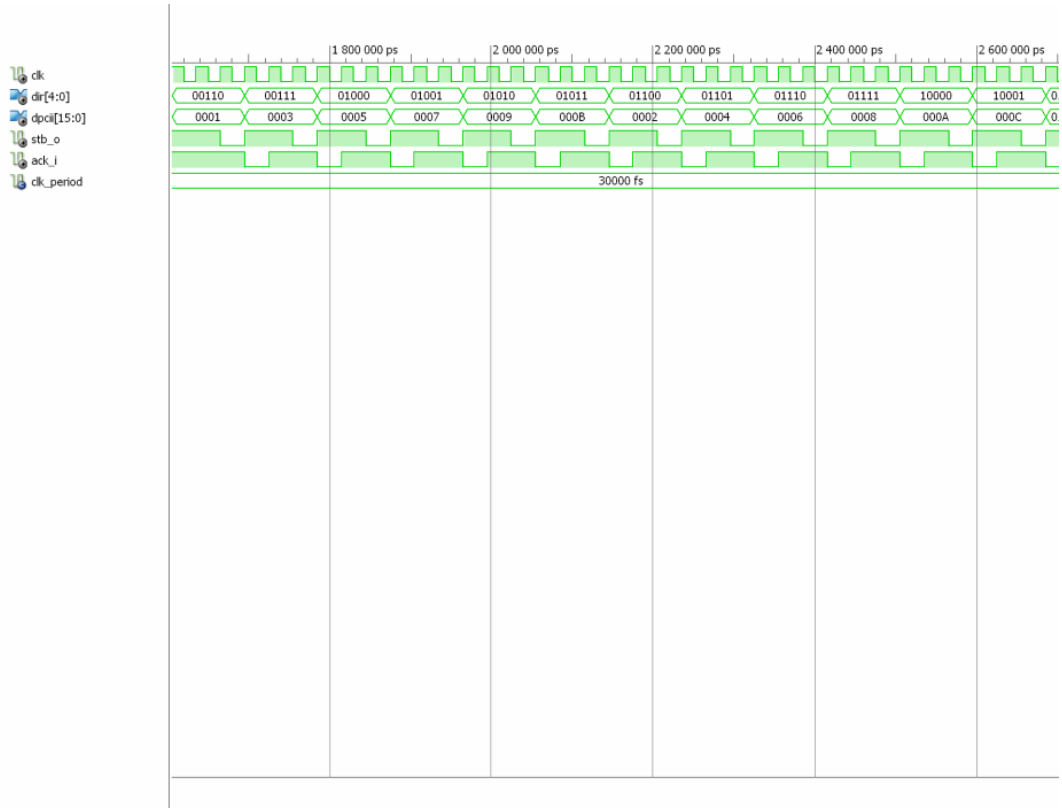


Figura 27: Lectura de los registros.

Todo esto se ha realizado obteniendo el resultado esperado y validando por lo tanto el diseño realizado.

Capítulo 4 CONCLUSIONES

Aunque el proyecto ha resultado ser largo y complejo el resultado final ha sido satisfactorio ya que se han alcanzado en gran medida los objetivos impuestos.

El diseño del PCB resulto complicado debido sobre todo al uso de componentes con pequeñas separación de pines lo que provocaba fuertes concentraciones de pistas en el rutado y nos obligo a utilizar una clase alta.

En el diseño de la FPGA el simulador nos dio múltiples problemas difíciles de solucionar, ya que el sistema fallaba sin dar una idea del motivo.

A pesar de estos problemas se ha conseguido diseñar una tarjeta de adquisición de datos de 12 canales simultáneos con una resolución de 16bits por canal y con una velocidad de más de 450ksps, que es mucho más del mínimo buscado y que cumple con las necesidades de comunicación con el ordenador mediante el puerto PCI, además puesto que usa un bus estándar como es el bus wishbone es fácil de modificar para hacerlo compatible con otros tipos de puertos.

Lo único malo es que por falta de presupuesto no pudo ser finalmente fabricada y por tanto el prototipo, que aunque se halla simulado, no se ha testeado.

Capítulo 5 FUTUROS DESARROLLOS

Los posibles desarrollos futuros serian entre otros los siguientes:

_Construir y probar el prototipo diseñado para comprobar su validez.

_Mejorar algunos aspecto físicos del diseño del PCB.

_Crear un circuito más complejo para la FPGA tal vez desarrollando un sistema por interrupciones.

_Diseñar la posibilidad de comunicación entre los conversores y la FPGA mediante un protocolo serie.

_Diseño de un driver y de una interfaz grafica para la utilización del conversor desde el ordenador de una forma más amena.

BIBLIOGRAFÍA

[1] www.ti.com

[2] www.ni.com

[3] http://www.enterpoint.co.uk/moelbryn/raggedstone1.html

[4] http://es.farnell.com/

[5] http://www.eurocircuits.com/

[6] www.wikipedia.com

Parte II ESTUDIO ECONÓMICO

Capítulo 1 ESTUDIO ECONÓMICO

Como ya se ha comentado antes no existen en el mercado tarjetas que concuerden con las necesidades existentes y puesto que cada vez es más habitual el uso de ordenadores en los sistemas de control existe una posibilidad de negocio que justifica la realización de este proyecto.

Este proyecto sería viable de poner en marcha por que se trata de entrar en un mercado donde la demanda supera a la oferta, por lo que no tendríamos que luchar mucho con otros proveedores de este tipo de tecnología asentados en el sector ya que hay cuota de mercado suficiente para todos, además la inversión inicial no será muy grande lo que reduce el riesgo de la operación.

El producto se puede vender con un buen margen, lo que aporta una gran rentabilidad y la obtención de beneficios con la venta de pocas unidades.

Como se ha comentado anteriormente el sector de las tarjetas de medida, sobre todo las de medición simultánea, es un sector en auge con unas excelentes expectativas de negocio. Nos encontramos en el momento idóneo para invertir este proyecto ya que podemos aprovechar el momento de bonanza del sector para introducirnos en él y que dentro de un tiempo cuando se empiece a estabilizar aprovechar la curva de experiencia obtenida para mantenernos en el sector como un proveedor de referencia.

Parte III MANUAL DE USUARIO

Capítulo 1 MANUAL DE USUARIO

1.1 INSTALACIÓN DEL HARDWARE.

Para instalar el dispositivo tan solo hay que conectarlo a una conexión PCI estándar presente en la mayoría de ordenadores, no son necesarias más conexiones de tipo hardware.

1.2 DESARROLLO DEL SOFTWARE.

El usuario tendrá que desarrollar el software necesario para enviar y recibir señales mediante el puerto PCI teniendo en cuenta lo siguiente:

1.2.1 USO DE LOS REGISTROS.

Los registros accesibles desde el ordenador ocupan las direcciones de memoria 0 a 18. Los 6 primeros registros son registros de solo escritura (Tabla 12). Los registros de configuración deberán ser adecuadamente cargados con la información necesaria para realizar la conversión, si la información existente en los registros no es la correcta, el sistema podría comportarse de manera inadecuada. El registro TIMER guarda el tiempo durante el cual se llevara a cabo la conversión en modo continuo, si no se llevarán a cabo conversiones continuas el valor de este registro es irrelevante y no hace falta definirlo. El registro CONTROL se encarga de iniciar la conversión única así como la conversión continua, también se puede detener de manera brusca la conversión mediante una orden por este registro.

Registro	*Dirección*
Registro de configuración de la parte baja del conversor 1 (CR1.1)	0
Registro de configuración de la parte alta del conversor 1 (CR1.2)	1
Registro de configuración de la parte baja del conversor 2 (CR2.1)	2
Registro de configuración de la parte alta del conversor 2 (CR2.2)	3
Registro que almacena el tiempo de conversión (TIMER)	4
Registro de control (CONTROL)	5

Tabla 12: Registros de escritura

Los 13 últimos registros (Tabla 13) son de solo lectura, los doce primeros son cada uno de los doce canales de conversión, el registro BUSY, del cual solo nos interesa el ultimo bit, nos indica si la conversión a finalizada y podemos empezar a leer los demás registros.

Registro	*Dirección*
Datos del canal A1 del conversor 1 (A11)	6
Datos del canal A2 del conversor 1 (A12)	7
Datos del canal B1 del conversor 1 (B11)	8
Datos del canal B2 del conversor 1 (B12)	9
Datos del canal C1 del conversor 1 (C11)	10
Datos del canal C2 del conversor 1 (C12)	11
Datos del canal A1 del conversor 2 (A21)	12
Datos del canal A2 del conversor 2 (A22)	13
Datos del canal B1 del conversor 2 (B21)	14
Datos del canal B2 del conversor 2 (B22)	15
Datos del canal C1 del conversor 2 (C21)	16

Datos del canal C2 del conversor 2 (C22)	17
Estado del sistema (BUSY)	18

Tabla 13: Registros de lectura

1.2.2 PROCEDIMIENTO A SEGUIR.

Lo primero se enviará a los registros correspondientes la configuración de los conversores para la función deseada. Después, en el caso de quererse una conversión continua, se enviara el tiempo de conversión que se almacenará en el registro TIMER, teniendo en cuenta que el tiempo de conversión será el valor del registro multiplicado por 30 nanosegundos.

Después se enviará el comando de inicio por el registro CONTROL, puesto que el sistema funciona detectando los cambios en el registro CONTROL, se aconseja poner primero el registro a cero y luego enviar la orden que se quiere dar al sistema.

Hecho esto se tendrá que controlar el registro BUSY hasta que se ponga a cero, una vez este a cero, se podrá empezar a leer los datos de los conversores. En el modo continuo, puesto que el sistema no es capaz de almacenar más de un dato de cada canal, la lectura se tendrá que hacer lo más rápido posible y en orden ascendente de dirección del registro para no perder ningún dato.

Parte IV CÓDIGO FUENTE

Capítulo 1 TARJETA

1.1 REGISTRO.

```
----------------------------------------------------------------------------
---------
-- Company:
-- Engineer:
--
-- Create Date:    14:24:29 01/18/2011
-- Design Name:
-- Module Name:    Registro - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Registro is
port (
      Reset             : in std_logic;
      PCI_CLK     : in std_logic;
      Enable       : in std_logic;
      Entrada           : in std_logic_vector(15 downto 0);
      Salida       : out std_logic_vector(15 downto 0)
);
end Registro;

architecture Behavioral of Registro is

begin
      process (PCI_CLK, Reset, Entrada, Enable)
            begin
            if Reset = '1' then
                  Salida <= (others => '0');
```

```
            elsif (PCI_CLK'event and PCI_CLK = '1') then
                  if (Enable = '1') then
                        Salida <= Entrada;
                  end if;
            end if;
      end process;

end Behavioral;
```

1.2 TRIESTADO.

```
----------------------------------------------------------------------------
---------
-- Company:
-- Engineer:
--
-- Create Date:    14:24:29 01/18/2011
-- Design Name:
-- Module Name:    Triestado - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Triestado is
port (
      Enable       : in std_logic;
      Entrada            : in std_logic_vector(15 downto 0);
      Salida       : out std_logic_vector(15 downto 0)
);
end Triestado;

architecture Behavioral of Triestado is

begin
      process (Entrada, Enable)
            begin
                  if Enable = '1' then
                        Salida <= Entrada;
                  else
```

```
                        Salida <= (others => 'Z');
                end if;
        end process;

end Behavioral;
```

1.3 MULTIPLEXOR2

```
----------------------------------------------------------------------------
---------
-- Company:
-- Engineer:
--
-- Create Date:    14:24:29 01/18/2011
-- Design Name:
-- Module Name:    Multiplexor2 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Multiplexor2 is
port (
        Direccion    : in std_logic;
        Entrada1            : in std_logic_vector(15 downto 0);
        Entrada2            : in std_logic_vector(15 downto 0);
        Salida       : out std_logic_vector(15 downto 0)
);
end Multiplexor2;

architecture Behavioral of Multiplexor2 is

begin
        process (Direccion, Entrada1, Entrada2)
        begin
                case Direccion is
                        when '0' => Salida <= Entrada1;
                        when '1' => Salida <= Entrada2;
                        when others => null;
                end case;
```

```
      end process;

end Behavioral;
```

1.4 MULTIPLEXOR13

```
----------------------------------------------------------------------------
---------
-- Company:
-- Engineer:
--
-- Create Date:    14:24:29 01/18/2011
-- Design Name:
-- Module Name:    Multiplexor13 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Multiplexor13 is
port (
      Direccion    : in std_logic_vector(4 downto 0);
      Entrada1           : in std_logic_vector(15 downto 0);
      Entrada2           : in std_logic_vector(15 downto 0);
      Entrada3           : in std_logic_vector(15 downto 0);
      Entrada4           : in std_logic_vector(15 downto 0);
      Entrada5           : in std_logic_vector(15 downto 0);
      Entrada6           : in std_logic_vector(15 downto 0);
      Entrada7           : in std_logic_vector(15 downto 0);
      Entrada8           : in std_logic_vector(15 downto 0);
      Entrada9           : in std_logic_vector(15 downto 0);
      Entrada10    : in std_logic_vector(15 downto 0);
      Entrada11    : in std_logic_vector(15 downto 0);
      Entrada12    : in std_logic_vector(15 downto 0);
      Entrada13    : in std_logic_vector(15 downto 0);
      Salida       : out std_logic_vector(15 downto 0)
);
end Multiplexor13;

architecture Behavioral of Multiplexor13 is
```

```
begin
	process (Direccion, Entrada1, Entrada2, Entrada3, Entrada4,
Entrada5, Entrada6, Entrada7, Entrada8, Entrada9, Entrada10, Entrada11,
Entrada12, Entrada13)
	begin
		case Direccion is
			when "00110" => Salida <= Entrada1;
			when "00111" => Salida <= Entrada2;
			when "01000" => Salida <= Entrada3;
			when "01001" => Salida <= Entrada4;
			when "01010" => Salida <= Entrada5;
			when "01011" => Salida <= Entrada6;
			when "01100" => Salida <= Entrada7;
			when "01101" => Salida <= Entrada8;
			when "01110" => Salida <= Entrada9;
			when "01111" => Salida <= Entrada10;
			when "10000" => Salida <= Entrada11;
			when "10001" => Salida <= Entrada12;
			when "10010" => Salida <= Entrada13;
			when others => Salida <= (others => '0');
		end case;
	end process;

end Behavioral;
```

1.5 DEMULTIPLEXOR6

```
----------------------------------------------------------------------------
---------
-- Company:
-- Engineer:
--
-- Create Date:    14:24:29 01/18/2011
-- Design Name:
-- Module Name:    Demultiplexor6 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity Demultiplexor6 is
port (
       Direccion    : in std_logic_vector(4 downto 0);
       Salida1            : out std_logic;
       Salida2            : out std_logic;
       Salida3            : out std_logic;
       Salida4            : out std_logic;
       Salida5            : out std_logic;
       Salida6            : out std_logic
);
end Demultiplexor6;

architecture Behavioral of Demultiplexor6 is

begin
       process (Direccion)
       begin
              Salida1 <= '0';
              Salida2 <= '0';
              Salida3 <= '0';
              Salida4 <= '0';
              Salida5 <= '0';
              Salida6 <= '0';
              case Direccion is
                     when "00000" => Salida1 <= '1';
                     when "00001" => Salida2 <= '1';
                     when "00010" => Salida3 <= '1';
                     when "00011" => Salida4 <= '1';
                     when "00100" => Salida5 <= '1';
                     when "00101" => Salida6 <= '1';
                     when others => null;
              end case;
       end process;

end Behavioral;
```

1.6 CONTROLADOR

```
----------------------------------------------------------------------------
---------
-- Company:
-- Engineer:
--
-- Create Date:    14:24:29 01/18/2011
-- Design Name:
-- Module Name:    Controlador - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
```

```
----------------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Controlador is
port (
       Reset               : in std_logic;
       PCI_CLK     : in std_logic;
       Timer               : in std_logic_vector(15 downto 0);
       Control             : in std_logic_vector(15 downto 0);
       BUSY1               : in std_logic;
       BUSY2               : in std_logic;
       EnableA11    : out std_logic;
       EnableA12    : out std_logic;
       EnableA21    : out std_logic;
       EnableA22    : out std_logic;
       EnableB11    : out std_logic;
       EnableB12    : out std_logic;
       EnableB21    : out std_logic;
       EnableB22    : out std_logic;
       EnableC11    : out std_logic;
       EnableC12    : out std_logic;
       EnableC21    : out std_logic;
       EnableC22    : out std_logic;
       EnableBusy   : out std_logic;
       EnableTri1   : out std_logic;
       EnableTri2   : out std_logic;
       DirCon1             : out std_logic;
       DirCon2             : out std_logic;
       PARSER1             : out  std_logic;
       PARSER2             : out  std_logic;
       HWSW1               : out std_logic;
       HWSW2               : out std_logic;
       REFEN1       : out std_logic;
       REFEN2       : out std_logic;
       CS1                 : out std_logic;
       CS2                 : out std_logic;
       RD1                 : out std_logic;
       RD2                 : out std_logic;
       CONVST_A1    : out std_logic;
       CONVST_A2    : out std_logic;
       CONVST_B1    : out std_logic;
       CONVST_B2    : out std_logic;
       CONVST_C1    : out std_logic;
       CONVST_C2    : out std_logic;
       STBY1               : out std_logic;
       STBY2               : out std_logic;
       RANGO1       : out std_logic;
       RANGO2       : out std_logic;
       RESET1       : out std_logic;
       RESET2       : out std_logic;
       WORDBYTE1    : out std_logic;
       WORDBYTE2    : out std_logic;
       Busy                : out std_logic_vector(15 downto 0)
);
```

```
end Controlador;

architecture Behavioral of Controlador is

	signal Proceso                          : std_logic;
	signal Continuo                  : std_logic;
	signal Inicia                    : std_logic;
	signal FinInicia                 : std_logic;
	signal Conver                    : std_logic;
	signal FinConver                 : std_logic;
	type    EstadosPrincipal_type    is    (Espera,    Inicializacion,
Conversion);
	signal estadopri, estadopri_sig: EstadosPrincipal_type;
	type EstadosInicializacion_type is (Espera1, Primera, Intermedio,
Segunda);
	signal estadoini, estadoini_sig: EstadosInicializacion_type;
	type EstadosConversion_type is (Cero, Uno, Dos, Tres, Cuatro);
	signal estadocon, estadocon_sig: EstadosConversion_type;
	signal Contador                  : integer range 1 to 6;
	signal Anterior                  : std_logic_vector(15 downto 0);
	signal Tiempo                    : std_logic_vector(15 downto 0) :=
"0000000000000000";
	signal InicioTiempo       : std_logic;

begin

	PARSER1 <= '0';
	PARSER2 <= '0';
	WORDBYTE1 <= '0';
	WORDBYTE2 <= '0';
	HWSW1 <= '1';
	HWSW2 <= '1';
	STBY1 <= '1';
	STBY2 <= '1';
	RANGO1 <= '0';
	RANGO2 <= '0';

	process    (PCI_CLK,    reset,    estadopri_sig,    estadoini_sig,
estadocon_sig, control)
	begin
		if (PCI_CLK'event and PCI_CLK = '1') then
			if (Reset = '1' or Control(2) = '1') then
				estadopri <= Espera;
				estadoini <= Espera1;
				estadocon <= Cero;
				reset1<= '1';
				reset2<= '1';
			else
				estadopri <= estadopri_sig;
				estadoini <= estadoini_sig;
				estadocon <= estadocon_sig;
				reset1<= '0';
				reset2<= '0';
			end if;
		end if;
	end process;

	process (PCI_CLK, InicioTiempo)
	begin
		if (PCI_CLK'event and PCI_CLK = '1') then
			if (InicioTiempo = '1') then
				Tiempo <= "0000000000000000";
			else
```

```
                Tiempo<= Tiempo + '1';
            end if;
        end if;
end process;

process (Control, PCI_CLK, Anterior, Continuo, Timer, Tiempo)
begin
        if Reset = '1' then
            Proceso <= '0';
            Continuo <= '0';
            InicioTiempo <= '0';
        elsif rising_edge(PCI_CLK) then
            if (Control(0) = '1' and Anterior(0) = '0') then
                Proceso <= '1';
                Continuo <= '0';
            elsif (Control(1) = '1' and Anterior(1) = '0') then
                Continuo <= '1';
                InicioTiempo <= '1';
            elsif (Continuo = '1') then
                Proceso <= '1';
                InicioTiempo <= '0';
                if ( Timer = Tiempo) then
                    Continuo <= '0';
                end if;
            else
                Proceso <= '0';
            end if;
            Anterior <= Control;
        end if;
end process;

process (Proceso, FinInicia, FinConver, estadopri)
begin
        estadopri_sig <= estadopri;
        case (estadopri) is
            when Espera =>
                if Proceso = '1' then
                    estadopri_sig <= Inicializacion;
                end if;
            when Inicializacion =>
                if FinInicia = '1' then
                    estadopri_sig <= Conversion;
                end if;
            when Conversion =>
                if FinConver = '1' then
                    estadopri_sig <= Espera;
                end if;
        end case;
end process;

process (estadopri)
begin
        EnableBusy <= '1';
        Busy <= (others => '0');
        Inicia <= '0';
        Conver <= '0';
        case estadopri is
            when Espera =>
                Busy(15) <= '0';
            when Inicializacion =>
                Inicia <= '1';
                Busy(15) <= '1';
            when Conversion =>
```

```
                    Conver <= '1';
                    Busy(15) <= '1';
        end case;
end process;

process (Inicia, estadoini)
begin
        estadoini_sig <= estadoini;
        case (estadoini) is
                when Espera1 =>
                        if (Inicia = '1') then
                                estadoini_sig <= Primera;
                        end if;
                when Primera =>
                        estadoini_sig <= Intermedio;
                when Intermedio =>
                        estadoini_sig <= Segunda;
                when Segunda =>
                        estadoini_sig <= Espera1;
        end case;
end process;

process (estadoini)
begin
        CS1 <= 'Z';
        CS2 <= 'Z';
        REFEN1 <= '0';
        REFEN2 <= '0';
        DirCon1 <= '0';
        DirCon2 <= '0';
        EnableTri1 <= '0';
        EnableTri2 <= '0';
        FinInicia <= '0';
        case (estadoini) is
                when Espera1 =>
                        null;
                when Primera =>
                        CS1 <= '1';
                        CS2 <= '1';
                        REFEN1 <= '1';
                        REFEN2 <= '1';
                        DirCon1 <= '1';
                        DirCon2 <= '1';
                        EnableTri1 <= '1';
                        EnableTri2 <= '1';
                when Intermedio =>
                        CS1 <= '1';
                        CS2 <= '1';
                        DirCon1 <= '1';
                        DirCon2 <= '1';
                        EnableTri1 <= '1';
                        EnableTri2 <= '1';
                when Segunda =>
                        CS1 <= '1';
                        CS2 <= '1';
                        REFEN1 <= '1';
                        REFEN2 <= '1';
                        EnableTri1 <= '1';
                        EnableTri2 <= '1';
                        FinInicia <= '1';
        end case;
end process;
```

```
process (Conver, BUSY1, BUSY2, estadocon)
begin
	estadocon_sig <= estadocon;
	case (estadocon) is
		when Cero =>
			if (Conver = '1') then
				estadocon_sig <= Uno;
			end if;
		when Uno =>
			if (BUSY1 = '0' and BUSY2 = '0') then
				estadocon_sig <= Dos;
				Contador <= 1;
			end if;
		when Dos =>
			if (Contador = 6) then
				estadocon_sig <= Cuatro;
			else
				estadocon_sig <= Tres;
			end if;
		when Tres =>
			estadocon_sig <= Dos;
			Contador <= Contador + 1;
		when Cuatro =>
				estadocon_sig <= Cero;
	end case;
end process;

process (estadocon)
begin
	CONVST_A1 <= '0';
	CONVST_A2 <= '0';
	CONVST_B1 <= '0';
	CONVST_B2 <= '0';
	CONVST_C1 <= '0';
	CONVST_C2 <= '0';
	CS1 <= 'Z';
	CS2 <= 'Z';
	RD1 <= '0';
	RD2 <= '0';
	EnableA11 <= '0';
	EnableA21 <= '0';
	EnableA12 <= '0';
	EnableA22 <= '0';
	EnableB11 <= '0';
	EnableB21 <= '0';
	EnableB12 <= '0';
	EnableB22 <= '0';
	EnableC11 <= '0';
	EnableC21 <= '0';
	EnableC12 <= '0';
	EnableC22 <= '0';
	FinConver <= '0';
	case estadocon is
		when Cero =>
			null;
		when Uno =>
			CONVST_A1 <= '1';
			CONVST_A2 <= '1';
			CONVST_B1 <= '1';
			CONVST_B2 <= '1';
			CONVST_C1 <= '1';
			CONVST_C2 <= '1';
		when Dos =>
```

```
                        CS1 <= '1';
                        CS2 <= '1';
                        RD1 <= '1';
                        RD2 <= '1';
                        CONVST_A1 <= '1';
                        CONVST_A2 <= '1';
                        CONVST_B1 <= '1';
                        CONVST_B2 <= '1';
                        CONVST_C1 <= '1';
                        CONVST_C2 <= '1';
                        case Contador is
                              when 1 =>
                                    EnableA11 <= '1';
                                    EnableA21 <= '1';
                              when 2 =>
                                    EnableA12 <= '1';
                                    EnableA22 <= '1';
                              when 3 =>
                                    EnableB11 <= '1';
                                    EnableB21 <= '1';
                              when 4 =>
                                    EnableB12 <= '1';
                                    EnableB22 <= '1';
                              when 5 =>
                                    EnableC11 <= '1';
                                    EnableC21 <= '1';
                              when 6 =>
                                    EnableC12 <= '1';
                                    EnableC22 <= '1';
                              when others => null;
                        end case;
                  when Tres =>
                        CS1 <= '1';
                        CS2 <= '1';
                        CONVST_A1 <= '1';
                        CONVST_A2 <= '1';
                        CONVST_B1 <= '1';
                        CONVST_B2 <= '1';
                        CONVST_C1 <= '1';
                        CONVST_C2 <= '1';
                  when Cuatro =>
                        CONVST_A1 <= '1';
                        CONVST_A2 <= '1';
                        CONVST_B1 <= '1';
                        CONVST_B2 <= '1';
                        CONVST_C1 <= '1';
                        CONVST_C2 <= '1';
                        FinConver <= '1';
            end case;
      end process;

end Behavioral;
```

1.7 OBSERVADOR

```
----------------------------------------------------------------------
---------
```

```
-- Company:
-- Engineer:
--
-- Create Date:    12:54:35 05/28/2011
-- Design Name:
-- Module Name:    Observador - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Observador is
port (
      stb          : in std_logic;
      PCI_CLK      : in std_logic;
      enable : out std_logic;
      ack          : out std_logic
);
end Observador;

architecture Behavioral of Observador is

      type Estados_type is (Espera, Valido);
      signal estado, estado_sig: Estados_type;

begin

      process (PCI_CLK, estado_sig)
      begin
            if rising_edge(PCI_CLK) then
                  estado <= estado_sig;
            end if;
      end process;

      process (stb, estado)
      begin
            estado_sig <= estado;
            case (estado) is
                  when Espera =>
                        if stb = '1' then
                              estado_sig <= Valido;
                        end if;
                  when Valido =>
                        if stb = '0' then
                              estado_sig <= Espera;
```

```
                        end if;
            end case;
      end process;

      process (estado)
      begin
            case (estado) is
                  when Espera =>
                        enable <= '0';
                        ack <= '0';
                  when Valido =>
                        enable <= '1';
                        ack <= '1';
            end case;
      end process;

end Behavioral;
```

Capítulo 2 SIMULACIONES

2.1 PRUREGISTRO

```
--------------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date:   18:15:22 03/02/2011
-- Design Name:
-- Module Name:   K:/Proyecto/Compartida/VHDL/Proyecto/pruRegistro.vhd
-- Project Name:  Proyecto
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: Registro
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
and
-- std_logic_vector for the ports of the unit under test.  Xilinx
recommends
-- that these types always be used for the top-level I/O of a design in
order
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
--------------------------------------------------------------------------------
-------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY pruRegistro IS
END pruRegistro;

ARCHITECTURE behavior OF pruRegistro IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Registro
    PORT(
```

```
         Reset : IN  std_logic;
         PCI_CLK : IN  std_logic;
         Enable : IN  std_logic;
         Entrada : IN  std_logic_vector(15 downto 0);
         Salida : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal Reset : std_logic := '0';
   signal PCI_CLK : std_logic := '0';
   signal Enable : std_logic := '0';
   signal Entrada : std_logic_vector(15 downto 0) := (others => '0');

      --Outputs
   signal Salida : std_logic_vector(15 downto 0);

      constant PCI_CLK_period : time := 30 ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: Registro PORT MAP (
         Reset => Reset,
         PCI_CLK => PCI_CLK,
         Enable => Enable,
         Entrada => Entrada,
         Salida => Salida
       );

   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name


   PCI_CLK_process :process
   begin
            PCI_CLK <= '0';
            wait for PCI_CLK_period/2;
            PCI_CLK <= '1';
            wait for PCI_CLK_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state.

      wait for PCI_CLK_period*5;

      -- insert stimulus here

            Enable <= '1';

            wait for PCI_CLK_period*5;

            Entrada <= "0000010100001010";

            wait for PCI_CLK_period*5;

            Enable <= '0';
```

```
            wait for PCI_CLK_period*5;

            Entrada <= "0000000000001111";

            wait for PCI_CLK_period*5;

            Reset <= '1';

            wait for PCI_CLK_period*5;

            Reset <= '0';

            wait for PCI_CLK_period*5;

            Enable <= '1';

            wait for PCI_CLK_period*5;

            Enable <= '0';

      wait;
   end process;

END;
```

2.2 *PRUTRIESTADO*

```
--------------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date:   18:45:12 03/02/2011
-- Design Name:
-- Module Name:   K:/Proyecto/Compartida/VHDL/Proyecto/pruTriestado.vhd
-- Project Name:  Proyecto
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: Triestado
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
and
-- std_logic_vector for the ports of the unit under test.  Xilinx
recommends
-- that these types always be used for the top-level I/O of a design in
order
```

```
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
--------------------------------------------------------------------------
-------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY pruTriestado IS
END pruTriestado;

ARCHITECTURE behavior OF pruTriestado IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Triestado
    PORT(
         Enable : IN  std_logic;
         Entrada : IN  std_logic_vector(15 downto 0);
         Salida : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal Enable : std_logic := '0';
   signal Entrada : std_logic_vector(15 downto 0) := (others => '0');

       --Outputs
   signal Salida : std_logic_vector(15 downto 0);

       constant PCI_CLK_period : time := 30ns;

BEGIN

       -- Instantiate the Unit Under Test (UUT)
   uut: Triestado PORT MAP (
          Enable => Enable,
          Entrada => Entrada,
          Salida => Salida
        );

   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state.

      wait for PCI_CLK_period*5;

      -- insert stimulus here

             Entrada <= "0000000000001111";

             wait for PCI_CLK_period*5;

             Enable <= '1';

             wait for PCI_CLK_period*5;

             Enable <= '0';
```

```
      wait;
   end process;

END;
```

2.3 PRUMUL2

```
--------------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date:   18:55:59 03/02/2011
-- Design Name:
-- Module Name:   K:/Proyecto/Compartida/VHDL/Proyecto/pruMul2.vhd
-- Project Name:  Proyecto
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: Multiplexor2
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
and
-- std_logic_vector for the ports of the unit under test.  Xilinx
recommends
-- that these types always be used for the top-level I/O of a design in
order
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
--------------------------------------------------------------------------------
-------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY pruMul2 IS
END pruMul2;

ARCHITECTURE behavior OF pruMul2 IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Multiplexor2
    PORT(
         Direccion : IN  std_logic;
         Entrada1 : IN  std_logic_vector(15 downto 0);
```

```
         Entrada2 : IN  std_logic_vector(15 downto 0);
         Salida : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal Direccion : std_logic := '0';
   signal Entrada1 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada2 : std_logic_vector(15 downto 0) := (others => '0');

      --Outputs
   signal Salida : std_logic_vector(15 downto 0);

      constant PCI_CLK_period : time := 30ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: Multiplexor2 PORT MAP (
          Direccion => Direccion,
          Entrada1 => Entrada1,
          Entrada2 => Entrada2,
          Salida => Salida
        );



   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state.

      wait for PCI_CLK_period*5;

      -- insert stimulus here

            Entrada1 <= "0000000011111111";
            Entrada2 <= "1111111100000000";

            wait for PCI_CLK_period*5;

            Direccion <= '0';

            wait for PCI_CLK_period*5;

            Direccion <= '1';

      wait;
   end process;

END;
```

2.4 PRUMUL13

```
--------------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date:   19:02:47 03/02/2011
-- Design Name:
-- Module Name:   K:/Proyecto/Compartida/VHDL/Proyecto/pruMul13.vhd
-- Project Name:  Proyecto
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: Multiplexor13
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
and
-- std_logic_vector for the ports of the unit under test.  Xilinx
recommends
-- that these types always be used for the top-level I/O of a design in
order
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
--------------------------------------------------------------------------------
-------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY pruMul13 IS
END pruMul13;

ARCHITECTURE behavior OF pruMul13 IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Multiplexor13
    PORT(
         Direccion : IN  std_logic_vector(4 downto 0);
         Entrada1 : IN  std_logic_vector(15 downto 0);
         Entrada2 : IN  std_logic_vector(15 downto 0);
         Entrada3 : IN  std_logic_vector(15 downto 0);
         Entrada4 : IN  std_logic_vector(15 downto 0);
         Entrada5 : IN  std_logic_vector(15 downto 0);
         Entrada6 : IN  std_logic_vector(15 downto 0);
         Entrada7 : IN  std_logic_vector(15 downto 0);
         Entrada8 : IN  std_logic_vector(15 downto 0);
         Entrada9 : IN  std_logic_vector(15 downto 0);
```

```
         Entrada10 : IN  std_logic_vector(15 downto 0);
         Entrada11 : IN  std_logic_vector(15 downto 0);
         Entrada12 : IN  std_logic_vector(15 downto 0);
         Entrada13 : IN  std_logic_vector(15 downto 0);
         Salida : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal Direccion : std_logic_vector(4 downto 0) := (others => '0');
   signal Entrada1 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada2 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada3 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada4 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada5 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada6 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada7 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada8 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada9 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada10 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada11 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada12 : std_logic_vector(15 downto 0) := (others => '0');
   signal Entrada13 : std_logic_vector(15 downto 0) := (others => '0');

      --Outputs
   signal Salida : std_logic_vector(15 downto 0);

      constant PCI_CLK_period : time := 30ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: Multiplexor13 PORT MAP (
         Direccion => Direccion,
         Entrada1 => Entrada1,
         Entrada2 => Entrada2,
         Entrada3 => Entrada3,
         Entrada4 => Entrada4,
         Entrada5 => Entrada5,
         Entrada6 => Entrada6,
         Entrada7 => Entrada7,
         Entrada8 => Entrada8,
         Entrada9 => Entrada9,
         Entrada10 => Entrada10,
         Entrada11 => Entrada11,
         Entrada12 => Entrada12,
         Entrada13 => Entrada13,
         Salida => Salida
        );


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100ms.

      wait for PCI_CLK_period*5;

      -- insert stimulus here

            Direccion <= "00000";
            Entrada1 <= "0000000000000001";
```

```
Entrada2 <= "0000000000000010";
Entrada3 <= "0000000000000100";
Entrada4 <= "0000000000001000";
Entrada5 <= "0000000000010000";
Entrada6 <= "0000000000100000";
Entrada7 <= "0000000001000000";
Entrada8 <= "0000000010000000";
Entrada9 <= "0000000100000000";
Entrada10 <= "0000001000000000";
Entrada11 <= "0000010000000000";
Entrada12 <= "0000100000000000";
Entrada13 <= "0001000000000000";

wait for PCI_CLK_period*5;

Direccion <= "00110";

wait for PCI_CLK_period*5;

Direccion <= "00111";

wait for PCI_CLK_period*5;

Direccion <= "01000";

wait for PCI_CLK_period*5;

Direccion <= "01001";

wait for PCI_CLK_period*5;

Direccion <= "01010";

wait for PCI_CLK_period*5;

Direccion <= "01011";

wait for PCI_CLK_period*5;

Direccion <= "01100";

wait for PCI_CLK_period*5;

Direccion <= "01101";

wait for PCI_CLK_period*5;

Direccion <= "01110";

wait for PCI_CLK_period*5;

Direccion <= "01111";

wait for PCI_CLK_period*5;

Direccion <= "10000";

wait for PCI_CLK_period*5;

Direccion <= "10001";

wait for PCI_CLK_period*5;
```

```
            Direccion <= "10010";

      wait;
   end process;

END;
```

2.5 *PRUDEMUL6*

```
--------------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date:   19:14:51 03/02/2011
-- Design Name:
-- Module Name:   K:/Proyecto/Compartida/VHDL/Proyecto/pruDemul6.vhd
-- Project Name:  Proyecto
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: Demultiplexor6
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
and
-- std_logic_vector for the ports of the unit under test.  Xilinx
recommends
-- that these types always be used for the top-level I/O of a design in
order
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
--------------------------------------------------------------------------------
-------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY pruDemul6 IS
END pruDemul6;

ARCHITECTURE behavior OF pruDemul6 IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Demultiplexor6
    PORT(
```

```
         Direccion : IN  std_logic_vector(4 downto 0);
         Salida1 : OUT  std_logic;
         Salida2 : OUT  std_logic;
         Salida3 : OUT  std_logic;
         Salida4 : OUT  std_logic;
         Salida5 : OUT  std_logic;
         Salida6 : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal Direccion : std_logic_vector(4 downto 0) := (others => '0');

      --Outputs
   signal Salida1 : std_logic;
   signal Salida2 : std_logic;
   signal Salida3 : std_logic;
   signal Salida4 : std_logic;
   signal Salida5 : std_logic;
   signal Salida6 : std_logic;

      constant PCI_CLK_period : time := 30ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: Demultiplexor6 PORT MAP (
         Direccion => Direccion,
         Salida1 => Salida1,
         Salida2 => Salida2,
         Salida3 => Salida3,
         Salida4 => Salida4,
         Salida5 => Salida5,
         Salida6 => Salida6
       );

   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100ms.

      wait for PCI_CLK_period*5;

      -- insert stimulus here


            wait for PCI_CLK_period*5;

            Direccion <= "00000";

            wait for PCI_CLK_period*5;

            Direccion <= "00001";

            wait for PCI_CLK_period*5;

            Direccion <= "00010";

            wait for PCI_CLK_period*5;

            Direccion <= "00011";
```

```
            wait for PCI_CLK_period*5;

            Direccion <= "00100";

            wait for PCI_CLK_period*5;

            Direccion <= "00101";

            wait for PCI_CLK_period*5;

            Direccion <= "00110";


      wait;
   end process;

END;
```

2.6 *PRUCONTROL*

```
--------------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date:   16:53:05 02/21/2011
-- Design Name:
-- Module Name:   K:/Proyecto/Compartida/VHDL/Proyecto/pruControl.vhd
-- Project Name:  Proyecto
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: Controlador
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
and
-- std_logic_vector for the ports of the unit under test.  Xilinx
recommends
-- that these types always be used for the top-level I/O of a design in
order
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
--------------------------------------------------------------------------------
-------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
```

```
USE ieee.numeric_std.ALL;

ENTITY pruControl IS
END pruControl;

ARCHITECTURE behavior OF pruControl IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Controlador
    PORT(
         Reset : IN  std_logic;
         PCI_CLK : IN  std_logic;
         Timer : IN  std_logic_vector(15 downto 0);
         Control : IN  std_logic_vector(15 downto 0);
         BUSY1 : IN  std_logic;
         BUSY2 : IN  std_logic;
         EnableA11 : OUT  std_logic;
         EnableA12 : OUT  std_logic;
         EnableA21 : OUT  std_logic;
         EnableA22 : OUT  std_logic;
         EnableB11 : OUT  std_logic;
         EnableB12 : OUT  std_logic;
         EnableB21 : OUT  std_logic;
         EnableB22 : OUT  std_logic;
         EnableC11 : OUT  std_logic;
         EnableC12 : OUT  std_logic;
         EnableC21 : OUT  std_logic;
         EnableC22 : OUT  std_logic;
         EnableBusy : OUT  std_logic;
         EnableTri1 : OUT  std_logic;
         EnableTri2 : OUT  std_logic;
         DirCon1 : OUT  std_logic;
         DirCon2 : OUT  std_logic;
         PARSER1 : OUT  std_logic;
         PARSER2 : OUT  std_logic;
         HWSW1 : OUT  std_logic;
         HWSW2 : OUT  std_logic;
         REFEN1 : OUT  std_logic;
         REFEN2 : OUT  std_logic;
         CS1 : OUT  std_logic;
         CS2 : OUT  std_logic;
         RD1 : OUT  std_logic;
         RD2 : OUT  std_logic;
         CONVST_A1 : OUT  std_logic;
         CONVST_A2 : OUT  std_logic;
         CONVST_B1 : OUT  std_logic;
         CONVST_B2 : OUT  std_logic;
         CONVST_C1 : OUT  std_logic;
         CONVST_C2 : OUT  std_logic;
         STBY1 : OUT  std_logic;
         STBY2 : OUT  std_logic;
         RANGO1 : OUT  std_logic;
         RANGO2 : OUT  std_logic;
         RESET1 : OUT  std_logic;
         RESET2 : OUT  std_logic;
         WORDBYTE1 : OUT  std_logic;
         WORDBYTE2 : OUT  std_logic;
         Busy : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;
```

```
   --Inputs
   signal Reset : std_logic := '0';
   signal PCI_CLK : std_logic := '0';
   signal Timer : std_logic_vector(15 downto 0) := (others => '0');
   signal Control : std_logic_vector(15 downto 0) := (others => '0');
   signal BUSY1 : std_logic := '0';
   signal BUSY2 : std_logic := '0';

      --Outputs
   signal EnableA11 : std_logic;
   signal EnableA12 : std_logic;
   signal EnableA21 : std_logic;
   signal EnableA22 : std_logic;
   signal EnableB11 : std_logic;
   signal EnableB12 : std_logic;
   signal EnableB21 : std_logic;
   signal EnableB22 : std_logic;
   signal EnableC11 : std_logic;
   signal EnableC12 : std_logic;
   signal EnableC21 : std_logic;
   signal EnableC22 : std_logic;
   signal EnableBusy : std_logic;
   signal EnableTri1 : std_logic;
   signal EnableTri2 : std_logic;
   signal DirCon1 : std_logic;
   signal DirCon2 : std_logic;
   signal PARSER1 : std_logic;
   signal PARSER2 : std_logic;
   signal HWSW1 : std_logic;
   signal HWSW2 : std_logic;
   signal REFEN1 : std_logic;
   signal REFEN2 : std_logic;
   signal CS1 : std_logic;
   signal CS2 : std_logic;
   signal RD1 : std_logic;
   signal RD2 : std_logic;
   signal CONVST_A1 : std_logic;
   signal CONVST_A2 : std_logic;
   signal CONVST_B1 : std_logic;
   signal CONVST_B2 : std_logic;
   signal CONVST_C1 : std_logic;
   signal CONVST_C2 : std_logic;
   signal STBY1 : std_logic;
   signal STBY2 : std_logic;
   signal RANGO1 : std_logic;
   signal RANGO2 : std_logic;
   signal RESET1 : std_logic;
   signal RESET2 : std_logic;
   signal WORDBYTE1 : std_logic;
   signal WORDBYTE2 : std_logic;
   signal Busy : std_logic_vector(15 downto 0);

      constant PCI_CLK_period : time := 30 ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: Controlador PORT MAP (
          Reset => Reset,
          PCI_CLK => PCI_CLK,
          Timer => Timer,
          Control => Control,
          BUSY1 => BUSY1,
```

```
        BUSY2 => BUSY2,
        EnableA11 => EnableA11,
        EnableA12 => EnableA12,
        EnableA21 => EnableA21,
        EnableA22 => EnableA22,
        EnableB11 => EnableB11,
        EnableB12 => EnableB12,
        EnableB21 => EnableB21,
        EnableB22 => EnableB22,
        EnableC11 => EnableC11,
        EnableC12 => EnableC12,
        EnableC21 => EnableC21,
        EnableC22 => EnableC22,
        EnableBusy => EnableBusy,
        EnableTri1 => EnableTri1,
        EnableTri2 => EnableTri2,
        DirCon1 => DirCon1,
        DirCon2 => DirCon2,
        PARSER1 => PARSER1,
        PARSER2 => PARSER2,
        HWSW1 => HWSW1,
        HWSW2 => HWSW2,
        REFEN1 => REFEN1,
        REFEN2 => REFEN2,
        CS1 => CS1,
        CS2 => CS2,
        RD1 => RD1,
        RD2 => RD2,
        CONVST_A1 => CONVST_A1,
        CONVST_A2 => CONVST_A2,
        CONVST_B1 => CONVST_B1,
        CONVST_B2 => CONVST_B2,
        CONVST_C1 => CONVST_C1,
        CONVST_C2 => CONVST_C2,
        STBY1 => STBY1,
        STBY2 => STBY2,
        RANGO1 => RANGO1,
        RANGO2 => RANGO2,
        RESET1 => RESET1,
        RESET2 => RESET2,
        WORDBYTE1 => WORDBYTE1,
        WORDBYTE2 => WORDBYTE2,
        Busy => Busy
      );

-- No clocks detected in port list. Replace <clock> below with
-- appropriate port name


PCI_CLK_process :process
begin
          PCI_CLK <= '0';
          wait for PCI_CLK_period/2;
          PCI_CLK <= '1';
          wait for PCI_CLK_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state.
```

```
wait for PCI_CLK_period*2;

-- insert stimulus here

        Control(0)<= '1';

        wait for PCI_CLK_period*6;

        BUSY1 <= '1';
        BUSY2 <= '1';

        wait for PCI_CLK_period*2;

        BUSY1 <= '0';
        BUSY2 <= '0';

        wait for PCI_CLK_period*14;

        Control(0)<= '0';

        wait for PCI_CLK_period*2;

        Control(0)<= '1';

        wait for PCI_CLK_period*6;

        reset<= '1';

        wait for PCI_CLK_period*2;

        reset<= '0';
        Control(0)<= '0';

        wait for PCI_CLK_period*2;

        Control(0)<= '1';

        wait for PCI_CLK_period*6;

        Control(2)<= '1';
        Control(0)<= '0';

        wait for PCI_CLK_period*2;

        Control(0)<= '1';
        Control(2)<= '0';

        wait for PCI_CLK_period*6;

        BUSY1 <= '1';
        BUSY2 <= '1';

        wait for PCI_CLK_period*2;

        BUSY1 <= '0';
        BUSY2 <= '0';

        wait for PCI_CLK_period*14;

        Timer <= "0000000000100011";
        Control(0) <= '0';
        Control(1) <= '1';
```

```
            wait for PCI_CLK_period*7;

            BUSY1 <= '1';
            BUSY2 <= '1';

            wait for PCI_CLK_period*2;

            BUSY1 <= '0';
            BUSY2 <= '0';

            wait for PCI_CLK_period*19;

            BUSY1 <= '1';
            BUSY2 <= '1';

            wait for PCI_CLK_period*2;

            BUSY1 <= '0';
            BUSY2 <= '0';

      wait;
   end process;

END;
```

2.7 *PRUOBSERVADOR*

```
--------------------------------------------------------------------------------
-------
-- Company:
-- Engineer:
--
-- Create Date:   18:06:56 05/28/2011
-- Design Name:
--                                    Module                                    Name:
C:/Juan/Proyecto/Compartida/VHDL/Proyecto/pruObservador.vhd
-- Project Name:  Proyecto
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: Observador
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
and
--  std_logic_vector  for  the  ports  of  the  unit  under  test.   Xilinx
recommends
-- that these types always be used for the top-level I/O of a design in
order
```

```
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
--------------------------------------------------------------------------
-------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY pruObservador IS
END pruObservador;

ARCHITECTURE behavior OF pruObservador IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Observador
    PORT(
         stb : IN  std_logic;
         PCI_CLK : IN  std_logic;
         enable : OUT  std_logic;
         ack : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal stb : std_logic := '0';
   signal PCI_CLK : std_logic := '0';

      --Outputs
   signal enable : std_logic;
   signal ack : std_logic;

      constant PCI_CLK_period : time := 30 ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: Observador PORT MAP (
          stb => stb,
          PCI_CLK => PCI_CLK,
          enable => enable,
          ack => ack
        );

   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name


   PCI_CLK_process :process
   begin
            PCI_CLK <= '1';
            wait for PCI_CLK_period/2;
            PCI_CLK <= '0';
            wait for PCI_CLK_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
```

```
      -- hold reset state.

      wait for PCI_CLK_period*2;

      -- insert stimulus here

            STB <= '1';

            WAIT FOR PCI_CLK_PERIOD*2;

            STB <= '0';

      wait;
   end process;

END;
```

2.8 CONJUNTO

```
--      Vhdl      test      bench      created      from      schematic
K:\Proyecto\Compartida\VHDL\Proyecto\Tarjeta.sch  -  Wed  Mar  23  18:58:47
2011
--
-- Notes:
--  1)  This  testbench  template  has  been  automatically  generated  using
types
-- std_logic and std_logic_vector for the ports of the unit under test.
-- Xilinx recommends that these types always be used for the top-level
-- I/O of a design in order to guarantee that the testbench will bind
-- correctly to the timing (post-route) simulation model.
-- 2) To use this template as your testbench, change the filename to any
-- name of your choice with the extension .vhd, and use the "Source->Add"
-- menu in Project Navigator to import the testbench. Then
-- edit the user defined section below, adding code to generate the
-- stimulus for your design.
--
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY Tarjeta_Tarjeta_sch_tb IS
END Tarjeta_Tarjeta_sch_tb;
ARCHITECTURE behavioral OF Tarjeta_Tarjeta_sch_tb IS

   COMPONENT Tarjeta
   PORT( CLK :      IN     STD_LOGIC;
         RESET     :      IN     STD_LOGIC;
         DPCIO     :      IN     STD_LOGIC_VECTOR (15 DOWNTO 0);
         DCON1     :      INOUT  STD_LOGIC_VECTOR (15 DOWNTO 0);
         DCON2     :      INOUT  STD_LOGIC_VECTOR (15 DOWNTO 0);
         DIR:      IN     STD_LOGIC_VECTOR (4 DOWNTO 0);
         DPCII     :      OUT    STD_LOGIC_VECTOR (15 DOWNTO 0);
         PS1:      OUT    STD_LOGIC;
         PS2:      OUT    STD_LOGIC;
         HS1:      OUT    STD_LOGIC;
         HS2:      OUT    STD_LOGIC;
         REF1      :      OUT    STD_LOGIC;
```

```
        REF2      :      OUT    STD_LOGIC;
        CS_1      :      OUT    STD_LOGIC;
        CS_2      :      OUT    STD_LOGIC;
        RD_1      :      OUT    STD_LOGIC;
        RD_2      :      OUT    STD_LOGIC;
        ST1:      OUT    STD_LOGIC;
        ST2:      OUT    STD_LOGIC;
        CONA1     :      OUT    STD_LOGIC;
        CONA2     :      OUT    STD_LOGIC;
        CONB1     :      OUT    STD_LOGIC;
        CONB2     :      OUT    STD_LOGIC;
        CONC1     :      OUT    STD_LOGIC;
        CONC2     :      OUT    STD_LOGIC;
        RAN1      :      OUT    STD_LOGIC;
        RAN2      :      OUT    STD_LOGIC;
        RES1      :      OUT    STD_LOGIC;
        RES2      :      OUT    STD_LOGIC;
        WB1:      OUT    STD_LOGIC;
        WB2:      OUT    STD_LOGIC;
        BU1:      IN     STD_LOGIC;
        BU2:      IN     STD_LOGIC;
                   STB_O : IN STD_LOGIC;
                   ACK_I : OUT STD_LOGIC);
END COMPONENT;

SIGNAL CLK:      STD_LOGIC;
SIGNAL RESET     :      STD_LOGIC;
SIGNAL DPCIO     :      STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL DCON1     :      STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL DCON2     :      STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL DIR:      STD_LOGIC_VECTOR (4 DOWNTO 0);
SIGNAL DPCII     :      STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL PS1:      STD_LOGIC;
SIGNAL PS2:      STD_LOGIC;
SIGNAL HS1:      STD_LOGIC;
SIGNAL HS2:      STD_LOGIC;
SIGNAL REF1      :      STD_LOGIC;
SIGNAL REF2      :      STD_LOGIC;
SIGNAL CS_1      :      STD_LOGIC;
SIGNAL CS_2      :      STD_LOGIC;
SIGNAL RD_1      :      STD_LOGIC;
SIGNAL RD_2      :      STD_LOGIC;
SIGNAL ST1:      STD_LOGIC;
SIGNAL ST2:      STD_LOGIC;
SIGNAL CONA1     :      STD_LOGIC;
SIGNAL CONA2     :      STD_LOGIC;
SIGNAL CONB1     :      STD_LOGIC;
SIGNAL CONB2     :      STD_LOGIC;
SIGNAL CONC1     :      STD_LOGIC;
SIGNAL CONC2     :      STD_LOGIC;
SIGNAL RAN1      :      STD_LOGIC;
SIGNAL RAN2      :      STD_LOGIC;
SIGNAL RES1      :      STD_LOGIC;
SIGNAL RES2      :      STD_LOGIC;
SIGNAL WB1:      STD_LOGIC;
SIGNAL WB2:      STD_LOGIC;
SIGNAL BU1:      STD_LOGIC;
SIGNAL BU2:      STD_LOGIC;
    SIGNAL STB_O : STD_LOGIC;
    SIGNAL ACK_I : STD_LOGIC;

    constant CLK_period : time := 30 ns;
```

```
BEGIN

   UUT: Tarjeta PORT MAP(
             CLK => CLK,
             RESET => RESET,
             DPCIO => DPCIO,
             DCON1 => DCON1,
             DCON2 => DCON2,
             DIR => DIR,
             DPCII => DPCII,
             PS1 => PS1,
             PS2 => PS2,
             HS1 => HS1,
             HS2 => HS2,
             REF1 => REF1,
             REF2 => REF2,
             CS_1 => CS_1,
             CS_2 => CS_2,
             RD_1 => RD_1,
             RD_2 => RD_2,
             ST1 => ST1,
             ST2 => ST2,
             CONA1 => CONA1,
             CONA2 => CONA2,
             CONB1 => CONB1,
             CONB2 => CONB2,
             CONC1 => CONC1,
             CONC2 => CONC2,
             RAN1 => RAN1,
             RAN2 => RAN2,
             RES1 => RES1,
             RES2 => RES2,
             WB1 => WB1,
             WB2 => WB2,
             BU1 => BU1,
             BU2 => BU2,
             STB_O => STB_O,
             ACK_I => ACK_I
   );

-- *** Test Bench - User Defined Section ***
   CLK_process : PROCESS
   begin
             CLK <= '0';
             wait for CLK_period/2;
             CLK <= '1';
             wait for CLK_period/2;
   end process;

       stim_proc: process
       begin

-- Reset del sistema

             bu1 <= '0';
             bu2 <= '0';
             reset <= '1';
             STB_O <= '0';
             WAIT FOR CLK_PERIOD/2;
             wait FOR CLK_period*2;

-- carga de la parte baja del registro del convertidor 1
```

```
                reset <= '0';
                DPCIO <= "0000001111111111";
                DIR <= "00000";
                DCON1 <= "ZZZZZZZZZZZZZZZZ";
                DCON2 <= "ZZZZZZZZZZZZZZZZ";
                STB_O <= '1';

                wait for CLK_period*2;

                STB_O <= '0';

                WAIT FOR CLK_PERIOD;

-- carga de la parte alta del registro del convertidor 1

                DPCIO <= "1110000000000000";
                DIR <= "00001";
                STB_O <= '1';

                wait for CLK_period*2;

                STB_O <= '0';

                WAIT FOR CLK_PERIOD;

-- carga de la parte baja del registro del convertidor 2

                DPCIO <= "0000001111111111";
                DIR <= "00010";
                STB_O <= '1';

                wait for CLK_period*2;

                STB_O <= '0';

                WAIT FOR CLK_PERIOD;

-- carga de la parte alta del registro del convertidor 2

                DPCIO <= "1110000000000000";
                DIR <= "00011";
                STB_O <= '1';

                wait for CLK_period*2;

                STB_O <= '0';

                WAIT FOR CLK_PERIOD;

-- Tiempo de ejecucion en modo continuo

                DPCIO <= "0000000000100011";
                DIR <= "00100";
                STB_O <= '1';

                wait for CLK_period*2;

                STB_O <= '0';

                WAIT FOR CLK_PERIOD;

-- inicio conversion unica
```

```
            DPCIO <= "0000000000000001";
            DIR <= "00101";
            STB_O <= '1';

            wait for CLK_period*2;

            STB_O <= '0';

            WAIT FOR CLK_PERIOD;

-- observando el registro busy

            DIR <= "10010";
            STB_O <= '1';

            wait until CONA1 = '1'; --esperar hasta que empieze la
conversion
            wait for CLK_period/2;

-- los conversores entan convirtiendo

            BU1 <= '1';
            BU2 <= '1';

            wait for CLK_period*12;

-- fin de la conversion de los conversores

            BU1 <= '0';
            BU2 <= '0';

            wait until RD_1 = '1';

-- recibiendo primer dato de ambos conversores

            DCON1 <= "0000000000000001";
            DCON2 <= "0000000000000010";

            wait until RD_1 = '0';

            DCON1 <= "0000000000000000";
            DCON2 <= "0000000000000000";

            wait until RD_1 = '1';

-- recibiendo segundo dato de ambos conversores

            DCON1 <= "0000000000000011";
            DCON2 <= "0000000000000100";

            wait until RD_1 = '0';

            DCON1 <= "0000000000000000";
            DCON2 <= "0000000000000000";

            wait until RD_1 = '1';

-- recibiendo tercer dato de ambos conversores

            DCON1 <= "0000000000000101";
            DCON2 <= "0000000000000110";

            wait until RD_1 = '0';
```

```
            DCON1 <= "0000000000000000";
            DCON2 <= "0000000000000000";

            wait until RD_1 = '1';

-- recibiendo cuarto dato de ambos conversores

            DCON1 <= "0000000000000111";
            DCON2 <= "0000000000001000";

            wait until RD_1 = '0';

            DCON1 <= "0000000000000000";
            DCON2 <= "0000000000000000";

            wait until RD_1 = '1';

-- recibiendo quinto dato de ambos conversores

            DCON1 <= "0000000000001001";
            DCON2 <= "0000000000001010";

            wait until RD_1 = '0';

            DCON1 <= "0000000000000000";
            DCON2 <= "0000000000000000";

            wait until RD_1 = '1';

-- recibiendo sexto dato de ambos conversores

            DCON1 <= "0000000000001011";
            DCON2 <= "0000000000001100";

            wait until RD_1 = '0';

            DCON1 <= "ZZZZZZZZZZZZZZZZ";
            DCON2 <= "ZZZZZZZZZZZZZZZZ";

            wait until DPCII(15) = '0'; --el sistema ha terminado la
conversion
            wait for CLK_period;
-- guardando el dato numero 1 por el PCI

            DIR <= "00110";
            STB_O <= '1';

            wait for CLK_period*2;
            assert DPCII = X"0001"
              report "Error en la lectura del registro 1"
              severity failure;

            STB_O <= '0';

            WAIT FOR CLK_PERIOD;

-- guardando el dato numero 2 por el PCI

            DIR <= "00111";
            STB_O <= '1';

            wait for CLK_period*2;
```

```
                STB_O <= '0';

                WAIT FOR CLK_PERIOD;

-- guardando el dato numero 3 por el PCI

                DIR <= "01000";
                STB_O <= '1';

                wait for CLK_period*2;

                STB_O <= '0';

                WAIT FOR CLK_PERIOD;


-- guardando el dato numero 4 por el PCI

                DIR <= "01001";
                STB_O <= '1';

                wait for CLK_period*2;

                STB_O <= '0';

                WAIT FOR CLK_PERIOD;


-- guardando el dato numero 5 por el PCI

                DIR <= "01010";
                STB_O <= '1';

                wait for CLK_period*2;

                STB_O <= '0';

                WAIT FOR CLK_PERIOD;


-- guardando el dato numero 6 por el PCI

                DIR <= "01011";
                STB_O <= '1';

                wait for CLK_period*2;

                STB_O <= '0';

                WAIT FOR CLK_PERIOD;


-- guardando el dato numero 7 por el PCI

                DIR <= "01100";
                STB_O <= '1';

                wait for CLK_period*2;

                STB_O <= '0';

                WAIT FOR CLK_PERIOD;
```

```
-- guardando el dato numero 8 por el PCI

            DIR <= "01101";
            STB_O <= '1';

            wait for CLK_period*2;

            STB_O <= '0';

            WAIT FOR CLK_PERIOD;


-- guardando el dato numero 9 por el PCI

            DIR <= "01110";
            STB_O <= '1';

            wait for CLK_period*2;

            STB_O <= '0';

            WAIT FOR CLK_PERIOD;


-- guardando el dato numero 10 por el PCI

            DIR <= "01111";
            STB_O <= '1';

            wait for CLK_period*2;

            STB_O <= '0';

            WAIT FOR CLK_PERIOD;


-- guardando el dato numero 11 por el PCI

            DIR <= "10000";
            STB_O <= '1';

            wait for CLK_period*2;

            STB_O <= '0';

            WAIT FOR CLK_PERIOD;


-- guardando el dato numero 12 por el PCI

            DIR <= "10001";
            STB_O <= '1';

            wait for CLK_period*2;

            STB_O <= '0';

            WAIT FOR CLK_PERIOD;


-- inicio conversion continua
```

```
DPCIO <= "0000000000000010";
DIR <= "00101";
STB_O <= '1';

wait for CLK_period*2;

STB_O <= '0';

WAIT FOR CLK_PERIOD;

DIR <= "10010";
STB_O <= '1';

wait until CONA1 = '1';
wait for CLK_period/2;

BU1 <= '1';
BU2 <= '1';

wait for CLK_period*12;

BU1 <= '0';
BU2 <= '0';

wait until RD_1 = '1';

DCON1 <= "0000000000000001";
DCON2 <= "0000000000000010";

wait until RD_1 = '0';

DCON1 <= "0000000000000000";
DCON2 <= "0000000000000000";

wait until RD_1 = '1';

DCON1 <= "0000000000000011";
DCON2 <= "0000000000000100";

wait until RD_1 = '0';

DCON1 <= "0000000000000000";
DCON2 <= "0000000000000000";

wait until RD_1 = '1';

DCON1 <= "0000000000000101";
DCON2 <= "0000000000000110";

wait until RD_1 = '0';

DCON1 <= "0000000000000000";
DCON2 <= "0000000000000000";

wait until RD_1 = '1';

DCON1 <= "0000000000000111";
DCON2 <= "0000000000001000";

wait until RD_1 = '0';

DCON1 <= "0000000000000000";
```

```
DCON2 <= "0000000000000000";

wait until RD_1 = '1';

DCON1 <= "0000000000001001";
DCON2 <= "0000000000001010";

wait until RD_1 = '0';

DCON1 <= "0000000000000000";
DCON2 <= "0000000000000000";

wait until RD_1 = '1';

DCON1 <= "0000000000001011";
DCON2 <= "0000000000001100";

wait until RD_1 = '0';

DCON1 <= "ZZZZZZZZZZZZZZZZ";
DCON2 <= "ZZZZZZZZZZZZZZZZ";

wait until DPCII(15) = '0';

DIR <= "00110";

wait for CLK_period;

DIR <= "00111";

wait for CLK_period;

DIR <= "01000";

wait for CLK_period;

DIR <= "01001";

wait for CLK_period;

DIR <= "01010";

wait for CLK_period;

DIR <= "01011";

WAIT FOR CLK_period/2;

BU1 <= '1';
BU2 <= '1';

wait for CLK_period/2;

DIR <= "01100";

wait for CLK_period;

DIR <= "01101";

wait for CLK_period;

DIR <= "01110";
```

```
wait for CLK_period;

DIR <= "01111";

wait for CLK_period;

DIR <= "10000";

wait for CLK_period;

DIR <= "10001";

wait for CLK_period;

DIR <= "10010";

wait for CLK_period*4.5;

BU1 <= '0';
BU2 <= '0';

wait until RD_1 = '1';

DCON1 <= "0000000000000001";
DCON2 <= "0000000000000010";

wait until RD_1 = '0';

DCON1 <= "0000000000000000";
DCON2 <= "0000000000000000";

wait until RD_1 = '1';

DCON1 <= "0000000000000011";
DCON2 <= "0000000000000100";

wait until RD_1 = '0';

DCON1 <= "0000000000000000";
DCON2 <= "0000000000000000";

wait until RD_1 = '1';

DCON1 <= "0000000000000101";
DCON2 <= "0000000000000110";

wait until RD_1 = '0';

DCON1 <= "0000000000000000";
DCON2 <= "0000000000000000";

wait until RD_1 = '1';

DCON1 <= "0000000000000111";
DCON2 <= "0000000000001000";

wait until RD_1 = '0';

DCON1 <= "0000000000000000";
DCON2 <= "0000000000000000";

wait until RD_1 = '1';
```

```
DCON1 <= "0000000000001001";
DCON2 <= "0000000000001010";

wait until RD_1 = '0';

DCON1 <= "0000000000000000";
DCON2 <= "0000000000000000";

wait until RD_1 = '1';

DCON1 <= "0000000000001011";
DCON2 <= "0000000000001100";

wait until RD_1 = '0';

DCON1 <= "ZZZZZZZZZZZZZZZZ";
DCON2 <= "ZZZZZZZZZZZZZZZZ";

wait until DPCII(15) = '0';

DIR <= "00110";

wait for CLK_period;

DIR <= "00111";

wait for CLK_period;

DIR <= "01000";

wait for CLK_period;

DIR <= "01001";

wait for CLK_period;

DIR <= "01010";

wait for CLK_period;

DIR <= "01011";

wait for CLK_period;

DIR <= "01100";

wait for CLK_period;

DIR <= "01101";

wait for CLK_period;

DIR <= "01110";

wait for CLK_period;

DIR <= "01111";

wait for CLK_period;

DIR <= "10000";

wait for CLK_period;
```

```
            DIR <= "10001";

            wait;

      END PROCESS;

-- *** End Test Bench - User Defined Section ***

END;
```

2.9 FINAL

```
--      Vhdl      test      bench      created      from      schematic
C:\Juan\Proyecto\Compartida\VHDL\Proyecto\tarjetapci.sch  -  Thu  Apr  28
17:31:22 2011
--
-- Notes:
--  1)  This  testbench  template  has  been  automatically  generated  using
types
-- std_logic and std_logic_vector for the ports of the unit under test.
-- Xilinx recommends that these types always be used for the top-level
-- I/O of a design in order to guarantee that the testbench will bind
-- correctly to the timing (post-route) simulation model.
-- 2) To use this template as your testbench, change the filename to any
-- name of your choice with the extension .vhd, and use the "Source->Add"
-- menu in Project Navigator to import the testbench. Then
-- edit the user defined section below, adding code to generate the
-- stimulus for your design.
--
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY tarjetapci_tarjetapci_sch_tb IS
END tarjetapci_tarjetapci_sch_tb;
ARCHITECTURE behavioral OF tarjetapci_tarjetapci_sch_tb IS

   COMPONENT tarjetapci
   PORT( direccion  :      OUT    STD_LOGIC_VECTOR (24 DOWNTO 0);
         DENTRADA  :      OUT    STD_LOGIC_VECTOR (15 DOWNTO 0);
         CLK_F     :      IN     STD_LOGIC;
         RESET_F   :      IN     STD_LOGIC;
         PS1_F     :      OUT    STD_LOGIC;
         PS2_F     :      OUT    STD_LOGIC;
         HS1_F     :      OUT    STD_LOGIC;
         HS2_F     :      OUT    STD_LOGIC;
         REF1_F    :      OUT    STD_LOGIC;
         REF2_F    :      OUT    STD_LOGIC;
         CS_1_F    :      OUT    STD_LOGIC;
         CS_2_F    :      OUT    STD_LOGIC;
         RD_1_F    :      OUT    STD_LOGIC;
         RD_2_F    :      OUT    STD_LOGIC;
         ST1_F     :      OUT    STD_LOGIC;
         ST2_F     :      OUT    STD_LOGIC;
         CONA1_F   :      OUT    STD_LOGIC;
```

```
        CONA2_F   :      OUT    STD_LOGIC;
        CONB1_F   :      OUT    STD_LOGIC;
        CONB2_F   :      OUT    STD_LOGIC;
        CONC1_F   :      OUT    STD_LOGIC;
        CONC2_F   :      OUT    STD_LOGIC;
        RAN1_F    :      OUT    STD_LOGIC;
        RAN2_F    :      OUT    STD_LOGIC;
        RES1_F    :      OUT    STD_LOGIC;
        RES2_F    :      OUT    STD_LOGIC;
        WB1_F     :      OUT    STD_LOGIC;
        WB2_F     :      OUT    STD_LOGIC;
        DCON1_F   :      INOUT  STD_LOGIC_VECTOR (15 DOWNTO 0);
        DCON2_F   :      INOUT  STD_LOGIC_VECTOR (15 DOWNTO 0);
        BU1_F     :      IN     STD_LOGIC;
        BU2_F     :      IN     STD_LOGIC;
        FRAME_F   :      IN     STD_LOGIC;
        IRDY_F    :      IN     STD_LOGIC;
        CBE_F     :      IN     STD_LOGIC_VECTOR (3 DOWNTO 0);
        AD_F      :      INOUT  STD_LOGIC_VECTOR (31 DOWNTO 0);
        TRDY_F    :      OUT    STD_LOGIC;
        DEVSEL_F  :      OUT    STD_LOGIC);
END COMPONENT;

SIGNAL direccion :      STD_LOGIC_VECTOR (24 DOWNTO 0);
SIGNAL DENTRADA  :      STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL CLK_F     :      STD_LOGIC;
SIGNAL RESET_F   :      STD_LOGIC;
SIGNAL PS1_F     :      STD_LOGIC;
SIGNAL PS2_F     :      STD_LOGIC;
SIGNAL HS1_F     :      STD_LOGIC;
SIGNAL HS2_F     :      STD_LOGIC;
SIGNAL REF1_F    :      STD_LOGIC;
SIGNAL REF2_F    :      STD_LOGIC;
SIGNAL CS_1_F    :      STD_LOGIC;
SIGNAL CS_2_F    :      STD_LOGIC;
SIGNAL RD_1_F    :      STD_LOGIC;
SIGNAL RD_2_F    :      STD_LOGIC;
SIGNAL ST1_F     :      STD_LOGIC;
SIGNAL ST2_F     :      STD_LOGIC;
SIGNAL CONA1_F   :      STD_LOGIC;
SIGNAL CONA2_F   :      STD_LOGIC;
SIGNAL CONB1_F   :      STD_LOGIC;
SIGNAL CONB2_F   :      STD_LOGIC;
SIGNAL CONC1_F   :      STD_LOGIC;
SIGNAL CONC2_F   :      STD_LOGIC;
SIGNAL RAN1_F    :      STD_LOGIC;
SIGNAL RAN2_F    :      STD_LOGIC;
SIGNAL RES1_F    :      STD_LOGIC;
SIGNAL RES2_F    :      STD_LOGIC;
SIGNAL WB1_F     :      STD_LOGIC;
SIGNAL WB2_F     :      STD_LOGIC;
SIGNAL DCON1_F   :      STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL DCON2_F   :      STD_LOGIC_VECTOR (15 DOWNTO 0);
SIGNAL BU1_F     :      STD_LOGIC;
SIGNAL BU2_F     :      STD_LOGIC;
SIGNAL FRAME_F   :      STD_LOGIC;
SIGNAL IRDY_F    :      STD_LOGIC;
SIGNAL CBE_F     :      STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL AD_F      :      STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL TRDY_F    :      STD_LOGIC;
SIGNAL DEVSEL_F  :      STD_LOGIC;

    constant CLK_period : time := 30 ns;
```

```
BEGIN

   UUT: tarjetapci PORT MAP(
             direccion => direccion,
             DENTRADA => DENTRADA,
             CLK_F => CLK_F,
             RESET_F => RESET_F,
             PS1_F => PS1_F,
             PS2_F => PS2_F,
             HS1_F => HS1_F,
             HS2_F => HS2_F,
             REF1_F => REF1_F,
             REF2_F => REF2_F,
             CS_1_F => CS_1_F,
             CS_2_F => CS_2_F,
             RD_1_F => RD_1_F,
             RD_2_F => RD_2_F,
             ST1_F => ST1_F,
             ST2_F => ST2_F,
             CONA1_F => CONA1_F,
             CONA2_F => CONA2_F,
             CONB1_F => CONB1_F,
             CONB2_F => CONB2_F,
             CONC1_F => CONC1_F,
             CONC2_F => CONC2_F,
             RAN1_F => RAN1_F,
             RAN2_F => RAN2_F,
             RES1_F => RES1_F,
             RES2_F => RES2_F,
             WB1_F => WB1_F,
             WB2_F => WB2_F,
             DCON1_F => DCON1_F,
             DCON2_F => DCON2_F,
             BU1_F => BU1_F,
             BU2_F => BU2_F,
             FRAME_F => FRAME_F,
             IRDY_F => IRDY_F,
             CBE_F => CBE_F,
             AD_F => AD_F,
             TRDY_F => TRDY_F,
             DEVSEL_F => DEVSEL_F
   );

-- *** Test Bench - User Defined Section ***
   CLK_process : PROCESS
    begin
             CLK_f <= '0';
             wait for CLK_period/2;
             CLK_f <= '1';
             wait for CLK_period/2;
   end process;

      stim_proc : process
      begin

             bu1_f <= '0';
             bu2_f <= '0';
             reset_f <= '1';
             frame_f <= '1';
             devsel_f <= '1';
             irdy_f <= '1';
             trdy_f <= '1';
```

```
wait FOR CLK_period*2;

reset_f <= '0';
frame_f <= '0';
ad_f <=x"00000000";
cbe_f <=x"7";
DCON1_F <="ZZZZZZZZZZZZZZZZ";
DCON2_F <="ZZZZZZZZZZZZZZZZ";

wait for CLK_period;

ad_f <=x"000003ff";
cbe_f <=x"f";
devsel_f <= '0';
irdy_f <= '0';
trdy_f <= '0';

WAIT FOR CLK_PERIOD;

frame_f <= '1';
devsel_f <= '1';
irdy_f <= '1';
trdy_f <= '1';

wait for clk_period;

frame_f <= '0';
ad_f <=x"00000001";
cbe_f <=x"7";

wait for clk_period;

ad_f <=x"0000e000";
cbe_f <=x"f";
devsel_f <= '0';
irdy_f <= '0';
trdy_f <= '0';

wait for clk_period;

frame_f <= '1';
devsel_f <= '1';
irdy_f <= '1';
trdy_f <= '1';

wait for clk_period;

frame_f <= '0';
ad_f <=x"00000002";
cbe_f <=x"7";

wait for clk_period;

ad_f <=x"000003ff";
cbe_f <=x"f";
devsel_f <= '0';
irdy_f <= '0';
trdy_f <= '0';

wait for clk_period;

frame_f <= '1';
```

```
devsel_f <= '1';
irdy_f <= '1';
trdy_f <= '1';

wait for clk_period;

frame_f <= '0';
ad_f <=x"00000003";
cbe_f <=x"7";

wait for clk_period;

ad_f <=x"0000e000";
cbe_f <=x"f";
devsel_f <= '0';
irdy_f <= '0';
trdy_f <= '0';

wait for clk_period;

frame_f <= '1';
devsel_f <= '1';
irdy_f <= '1';
trdy_f <= '1';

wait for clk_period;

frame_f <= '0';
ad_f <=x"00000004";
cbe_f <=x"7";

wait for clk_period;

ad_f <=x"00000023";
cbe_f <=x"f";
devsel_f <= '0';
irdy_f <= '0';
trdy_f <= '0';

wait for clk_period;

frame_f <= '1';
devsel_f <= '1';
irdy_f <= '1';
trdy_f <= '1';

wait for clk_period;

frame_f <= '0';
ad_f <=x"00000005";
cbe_f <=x"7";

wait for clk_period;

ad_f <=x"00000001";
cbe_f <=x"f";
devsel_f <= '0';
irdy_f <= '0';
trdy_f <= '0';

wait for clk_period;

frame_f <= '1';
```

```
                devsel_f <= '1';
                irdy_f <= '1';
                trdy_f <= '1';

                wait for clk_period;

                frame_f <= '0';
                ad_f <=x"00000012";
                cbe_f <=x"6";

                wait for clk_period;

                cbe_f <=x"f";
                devsel_f <= '0';
                irdy_f <= '0';
                trdy_f <= '0';

                wait until CONA1_F = '1';
                WAIT FOR CLK_PERIOD/2;

                BU1_F <= '1';
                BU2_F <= '1';

                wait for CLK_period*12;

-- fin de la conversion de los conversores

                BU1_F <= '0';
                BU2_F <= '0';

                wait until RD_1_F = '1';

-- recibiendo primer dato de ambos conversores

                DCON1_F <= "0000000000000001";
                DCON2_F <= "0000000000000010";

                wait until RD_1_F = '0';

                DCON1_F <= "0000000000000000";
                DCON2_F <= "0000000000000000";

                wait until RD_1_F = '1';

-- recibiendo segundo dato de ambos conversores

                DCON1_F <= "0000000000000011";
                DCON2_F <= "0000000000000100";

                wait until RD_1_F = '0';

                DCON1_F <= "0000000000000000";
                DCON2_F <= "0000000000000000";

                wait until RD_1_F = '1';

-- recibiendo tercer dato de ambos conversores

                DCON1_F <= "0000000000000101";
                DCON2_F <= "0000000000000110";

                wait until RD_1_F = '0';
```

```
                DCON1_F <= "0000000000000000";
                DCON2_F <= "0000000000000000";

                wait until RD_1_F = '1';

-- recibiendo cuarto dato de ambos conversores

                DCON1_F <= "0000000000000111";
                DCON2_F <= "0000000000001000";

                wait until RD_1_F = '0';

                DCON1_F <= "0000000000000000";
                DCON2_F <= "0000000000000000";

                wait until RD_1_F = '1';

-- recibiendo quinto dato de ambos conversores

                DCON1_F <= "0000000000001001";
                DCON2_F <= "0000000000001010";

                wait until RD_1_F = '0';

                DCON1_F <= "0000000000000000";
                DCON2_F <= "0000000000000000";

                wait until RD_1_F = '1';

-- recibiendo sexto dato de ambos conversores

                DCON1_F <= "0000000000001011";
                DCON2_F <= "0000000000001100";

                wait until RD_1_F = '0';

                DCON1_F <= "ZZZZZZZZZZZZZZZZ";
                DCON2_F <= "ZZZZZZZZZZZZZZZZ";

                wait until ad_f(15) = '0'; --el sistema ha terminado la
conversion
                wait for CLK_period;

-- guardando el dato numero 1 por el PCI

                frame_f <= '1';
                devsel_f <= '1';
                irdy_f <= '1';
                trdy_f <= '1';

                wait for clk_period;

                frame_f <= '0';
                ad_f <=x"00000006";
                cbe_f <=x"6";

                wait for CLK_period;

                cbe_f <=x"f";
                devsel_f <= '0';
                irdy_f <= '0';
                trdy_f <= '0';
```

```
            wait for CLK_period;

-- guardando el dato numero 2 por el PCI

            frame_f <= '1';
            devsel_f <= '1';
            irdy_f <= '1';
            trdy_f <= '1';

            wait for clk_period;

            frame_f <= '0';
            ad_f <=x"00000007";
            cbe_f <=x"6";

            wait for CLK_period;

            cbe_f <=x"f";
            devsel_f <= '0';
            irdy_f <= '0';
            trdy_f <= '0';

            wait for CLK_period;

-- guardando el dato numero 3 por el PCI

            frame_f <= '1';
            devsel_f <= '1';
            irdy_f <= '1';
            trdy_f <= '1';

            wait for clk_period;

            frame_f <= '0';
            ad_f <=x"00000008";
            cbe_f <=x"6";

            wait for CLK_period;

            cbe_f <=x"f";
            devsel_f <= '0';
            irdy_f <= '0';
            trdy_f <= '0';

            wait for CLK_period;

-- guardando el dato numero 4 por el PCI

            frame_f <= '1';
            devsel_f <= '1';
            irdy_f <= '1';
            trdy_f <= '1';

            wait for clk_period;

            frame_f <= '0';
            ad_f <=x"00000009";
            cbe_f <=x"6";

            wait for CLK_period;

            cbe_f <=x"f";
            devsel_f <= '0';
```

```
            irdy_f <= '0';
            trdy_f <= '0';

            wait for CLK_period;

-- guardando el dato numero 5 por el PCI

            frame_f <= '1';
            devsel_f <= '1';
            irdy_f <= '1';
            trdy_f <= '1';

            wait for clk_period;

            frame_f <= '0';
            ad_f <=x"0000000a";
            cbe_f <=x"6";

            wait for CLK_period;

            cbe_f <=x"f";
            devsel_f <= '0';
            irdy_f <= '0';
            trdy_f <= '0';

            wait for CLK_period;

-- guardando el dato numero 6 por el PCI

            frame_f <= '1';
            devsel_f <= '1';
            irdy_f <= '1';
            trdy_f <= '1';

            wait for clk_period;

            frame_f <= '0';
            ad_f <=x"0000000b";
            cbe_f <=x"6";

            wait for CLK_period;

            cbe_f <=x"f";
            devsel_f <= '0';
            irdy_f <= '0';
            trdy_f <= '0';

            wait for CLK_period;

-- guardando el dato numero 7 por el PCI

            frame_f <= '1';
            devsel_f <= '1';
            irdy_f <= '1';
            trdy_f <= '1';

            wait for clk_period;

            frame_f <= '0';
            ad_f <=x"0000000c";
            cbe_f <=x"6";

            wait for CLK_period;
```

```
                cbe_f <=x"f";
                devsel_f <= '0';
                irdy_f <= '0';
                trdy_f <= '0';

                wait for CLK_period;

-- guardando el dato numero 8 por el PCI

                frame_f <= '1';
                devsel_f <= '1';
                irdy_f <= '1';
                trdy_f <= '1';

                wait for clk_period;

                frame_f <= '0';
                ad_f <=x"0000000d";
                cbe_f <=x"6";

                wait for CLK_period;

                cbe_f <=x"f";
                devsel_f <= '0';
                irdy_f <= '0';
                trdy_f <= '0';

                wait for CLK_period;

-- guardando el dato numero 9 por el PCI

                frame_f <= '1';
                devsel_f <= '1';
                irdy_f <= '1';
                trdy_f <= '1';

                wait for clk_period;

                frame_f <= '0';
                ad_f <=x"0000000e";
                cbe_f <=x"6";

                wait for CLK_period;

                cbe_f <=x"f";
                devsel_f <= '0';
                irdy_f <= '0';
                trdy_f <= '0';

                wait for CLK_period;

-- guardando el dato numero 10 por el PCI

                frame_f <= '1';
                devsel_f <= '1';
                irdy_f <= '1';
                trdy_f <= '1';

                wait for clk_period;

                frame_f <= '0';
                ad_f <=x"0000000f";
```

```
              cbe_f <=x"6";

              wait for CLK_period;

              cbe_f <=x"f";
              devsel_f <= '0';
              irdy_f <= '0';
              trdy_f <= '0';

              wait for CLK_period;

-- guardando el dato numero 11 por el PCI

              frame_f <= '1';
              devsel_f <= '1';
              irdy_f <= '1';
              trdy_f <= '1';

              wait for clk_period;

              frame_f <= '0';
              ad_f <=x"00000010";
              cbe_f <=x"6";

              wait for CLK_period;

              cbe_f <=x"f";
              devsel_f <= '0';
              irdy_f <= '0';
              trdy_f <= '0';

              wait for CLK_period;

-- guardando el dato numero 12 por el PCI

              frame_f <= '1';
              devsel_f <= '1';
              irdy_f <= '1';
              trdy_f <= '1';

              wait for clk_period;

              frame_f <= '0';
              ad_f <=x"00000011";
              cbe_f <=x"6";

              wait for CLK_period;

              cbe_f <=x"f";
              devsel_f <= '0';
              irdy_f <= '0';
              trdy_f <= '0';

              wait for CLK_period;


       wait;
       end process;
-- *** End Test Bench - User Defined Section ***

END;
```

Parte V DATASHEETS

Capítulo 1 ADS8556

TEXAS INSTRUMENTS

ADS8556
ADS8557
ADS8558

www.ti.com

SBAS404A–OCTOBER 2006–REVISED AUGUST 2009

16-, 14-, 12-Bit, Six-Channel, Simultaneous Sampling ANALOG-TO-DIGITAL CONVERTERS

FEATURES

- Family of 16-, 14-, 12-Bit, Pin- and Software-Compatible ADCs
- Six SAR ADCs Grouped in Three Pairs
- Maximum Data Rate Per Channel with Internal Conversion Clock and Reference:
  ADS8556: 630kSPS (PAR) or 450kSPS (SER)
  ADS8557: 670kSPS (PAR) or 470kSPS (SER)
  ADS8558: 730kSPS (PAR) or 500kSPS (SER)
- Maximum Data Rate with External Conversion Clock and Reference:
  800kSPS (PAR) or 530kSPS (SER)
- Pin-Selectable or Programmable Input Voltage Ranges: Up to ±12V
- Excellent Signal-to-Noise Performance:
  91.5dB (ADS8556)
  85dB (ADS8557)
  73.9dB (ADS8558)
- Programmable and Buffered Internal Reference: 0.5V to 2.5V and 0.5V to 3.0V
- Comprehensive Power-Down Modes:
  Deep Power-Down (Standby Mode)
  Partial Power-Down
  Auto-Nap Power-Down
- Selectable Parallel or Serial Interface
- Operating Temperature Range:
  –40°C to +125°C
- LQFP-64 Package

APPLICATIONS

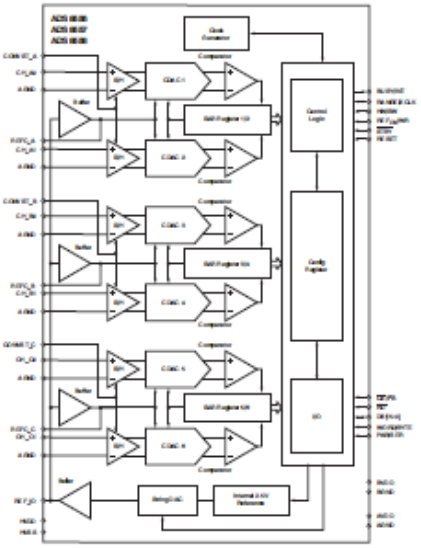
- Power Quality Measurement
- Protection Relays
- Multi-Axis Motor Control
- Programmable Logic Controllers
- Industrial Data Acquisition

DESCRIPTION

The ADS8556/7/8 contain six low-power, 16-, 14-, or 12-bit, successive approximation register (SAR) based analog-to-digital converters (ADCs) with true bipolar inputs. Each channel contains a sample-and-hold circuit that allows simultaneous high-speed multi-channel signal acquisition.

The ADS8556/7/8 support data rates of up to 730kSPS in parallel interface mode or up to 500kSPS if the serial interface is used. The bus width of the parallel interface can be set to eight or 16 bits. In serial mode, up to three output channels can be activated.

The ADS8556/7/8 is specified over the full industrial temperature range of –40°C to +125°C and is available in an LQFP-64 package.



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

All trademarks are the property of their respective owners.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of the Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Copyright © 2006–2009, Texas Instruments Incorporated

Capítulo 2 OPA2211

TEXAS INSTRUMENTS

OPA211
OPA2211

www.ti.com

SBOS377C–OCTOBER 2006–REVISED MAY 2009

1.1nV/√Hz Noise, Low Power, Precision Operational Amplifier in Small DFN-8 Package

FEATURES

- LOW VOLTAGE NOISE: 1.1nV/√Hz at 1kHz
- INPUT VOLTAGE NOISE: $80nV_{PP}$ (0.1Hz to 10Hz)
- THD+N: –136dB (G = 1, f = 1kHz)
- OFFSET VOLTAGE: 125µV (max)
- OFFSET VOLTAGE DRIFT: 0.35µV/°C (typ)
- LOW SUPPLY CURRENT: 3.6mA/Ch (typ)
- UNITY-GAIN STABLE
- GAIN BANDWIDTH PRODUCT: 80MHz (G = 100) 45MHz (G = 1)
- SLEW RATE: 27V/µs
- 16-BIT SETTLING: 700ns
- WIDE SUPPLY RANGE: ±2.25V to ±18V, +4.5V to +36V
- RAIL-TO-RAIL OUTPUT
- OUTPUT CURRENT: 30mA
- DFN-8 (3mm × 3mm), MSOP-8, AND SO-8

APPLICATIONS

- PLL LOOP FILTER
- LOW-NOISE, LOW-POWER SIGNAL PROCESSING
- 16-BIT ADC DRIVERS
- DAC OUTPUT AMPLIFIERS
- ACTIVE FILTERS
- LOW-NOISE INSTRUMENTATION AMPS
- ULTRASOUND AMPLIFIERS
- PROFESSIONAL AUDIO PREAMPLIFIERS
- LOW-NOISE FREQUENCY SYNTHESIZERS
- INFRARED DETECTOR AMPLIFIERS
- HYDROPHONE AMPLIFIERS
- GEOPHONE AMPLIFIERS
- MEDICAL

DESCRIPTION

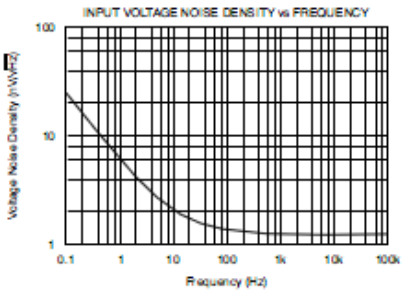
The OPA211 series of precision operational amplifiers achieves very low 1.1nV/√Hz noise density with a supply current of only 3.6mA. This series also offers rail-to-rail output swing, which maximizes dynamic range.

The extremely low voltage and low current noise, high speed, and wide output swing of the OPA211 series make these devices an excellent choice as a loop filter amplifier in PLL applications.

In precision data acquisition applications, the OPA211 series of op amps provides 700ns settling time to 16-bit accuracy throughout 10V output swings. This ac performance, combined with only 125µV of offset and 0.35µV/°C of drift over temperature, makes the OPA211 ideal for driving high-precision 16-bit analog-to-digital converters (ADCs) or buffering the output of high-resolution digital-to-analog converters (DACs).

The OPA211 series is specified over a wide dual-power supply range of ±2.25V to ±18V, or for single-supply operation from +4.5V to +36V.

The OPA211 is available in the small DFN-8 (3mm × 3mm), MSOP-8, and SO-8 packages. A dual version, the OPA2211, is available in the DFN-8 (3mm × 3mm) or an SO-8 PowerPAD™ package. This series of op amps is specified from T_A = –40°C to +125°C.



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

PowerPAD is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of the Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Copyright © 2006–2009, Texas Instruments Incorporated

Índice de planos

Parte I ***Esquema eléctrico*** ***126***

Capítulo 1 ***Esquema eléctrico*** ***127***

Parte II ***PCB*** ***128***

Capítulo 1 ***Capa superior*** ***129***

Capítulo 2 ***Capa inferior*** ***130***

Capítulo 3 ***Capa de masa*** ***131***

Capítulo 4 ***Capa de tensión*** ***132***

Capítulo 5 ***Componentes*** ***133***

Parte III ***FPGA*** ***134***

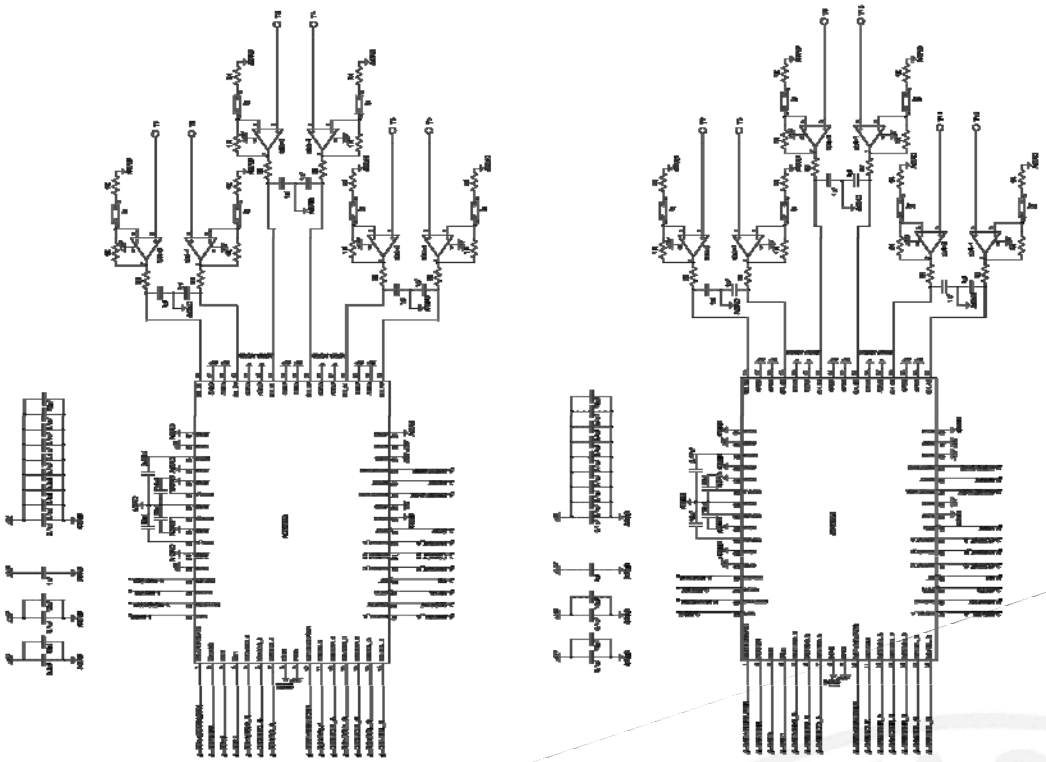
Capítulo 1 ***Tarjeta.sch*** ***135***

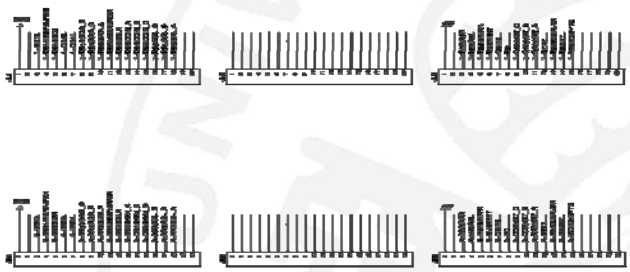
Capítulo 2 ***TarjetaPCI.sch*** ***136***

Parte I ESQUEMA

ELÉCTRICO

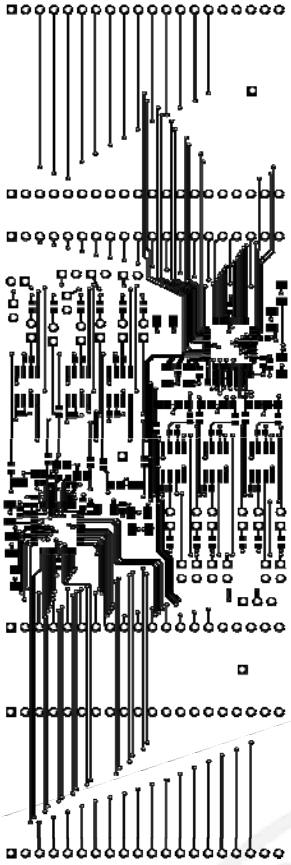
Capítulo 1 ESQUEMA ELÉCTRICO



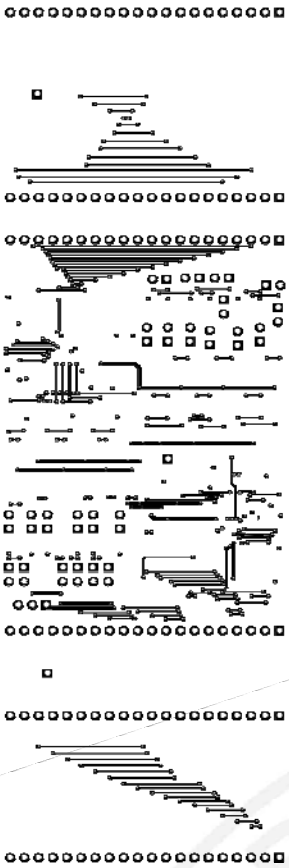


Parte II PCB

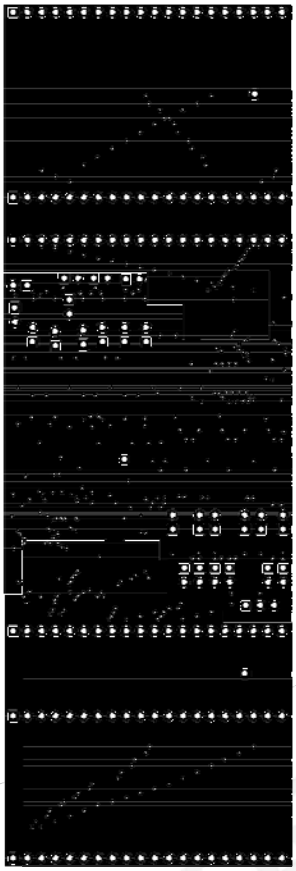
Capítulo 1 CAPA SUPERIOR



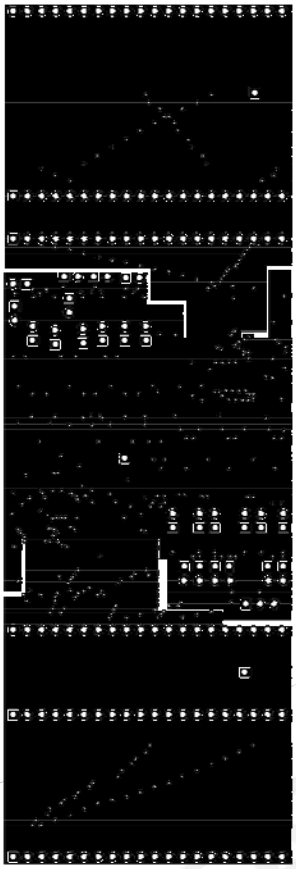
Capítulo 2 CAPA INFERIOR



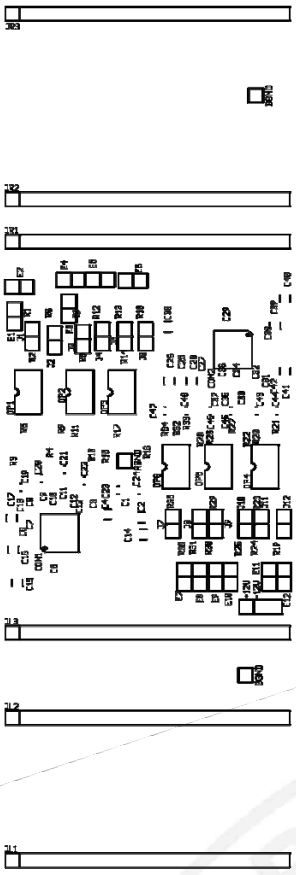
Capítulo 3 CAPA DE MASA



Capítulo 4 CAPA DE TENSIÓN

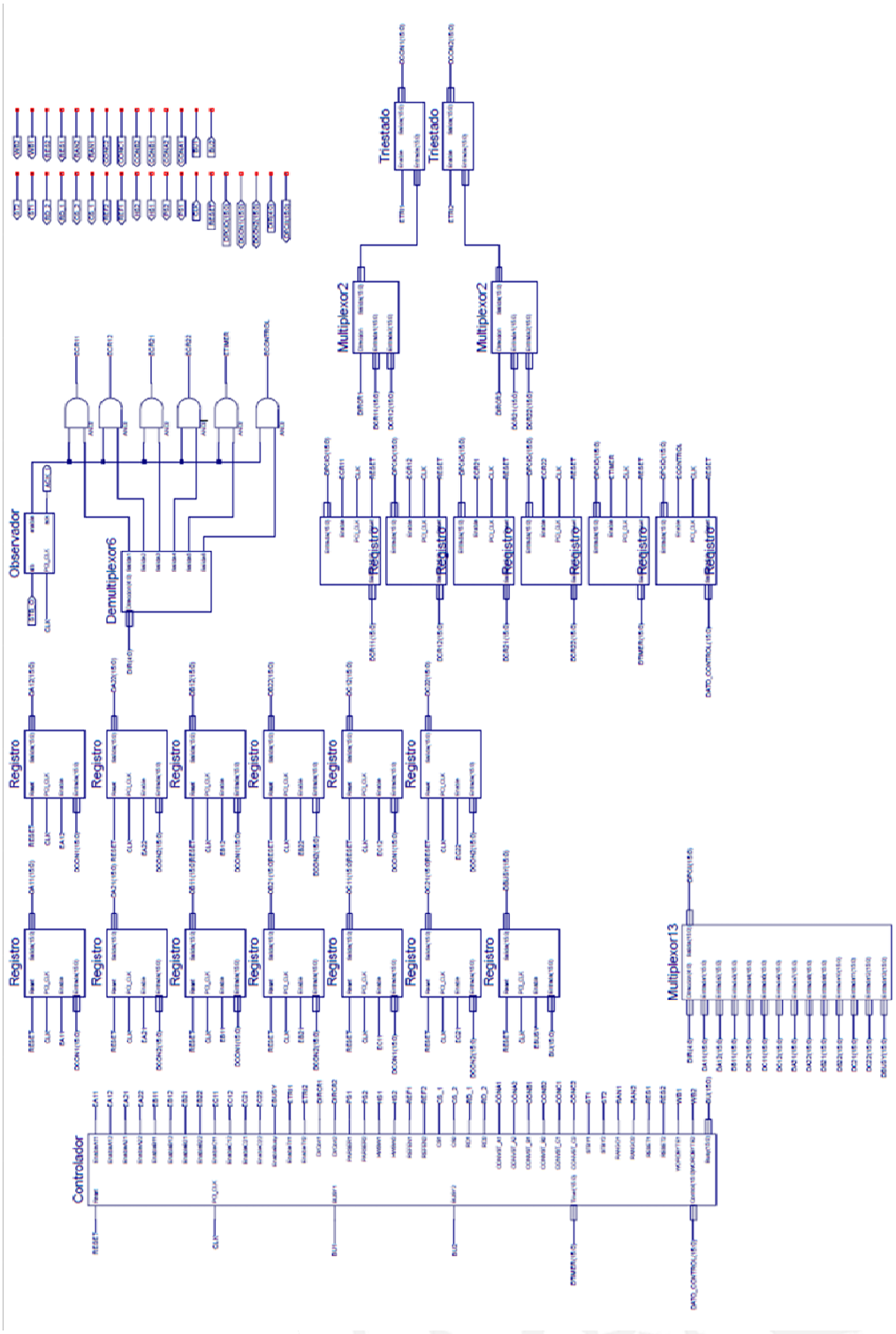


Capítulo 5 COMPONENTES

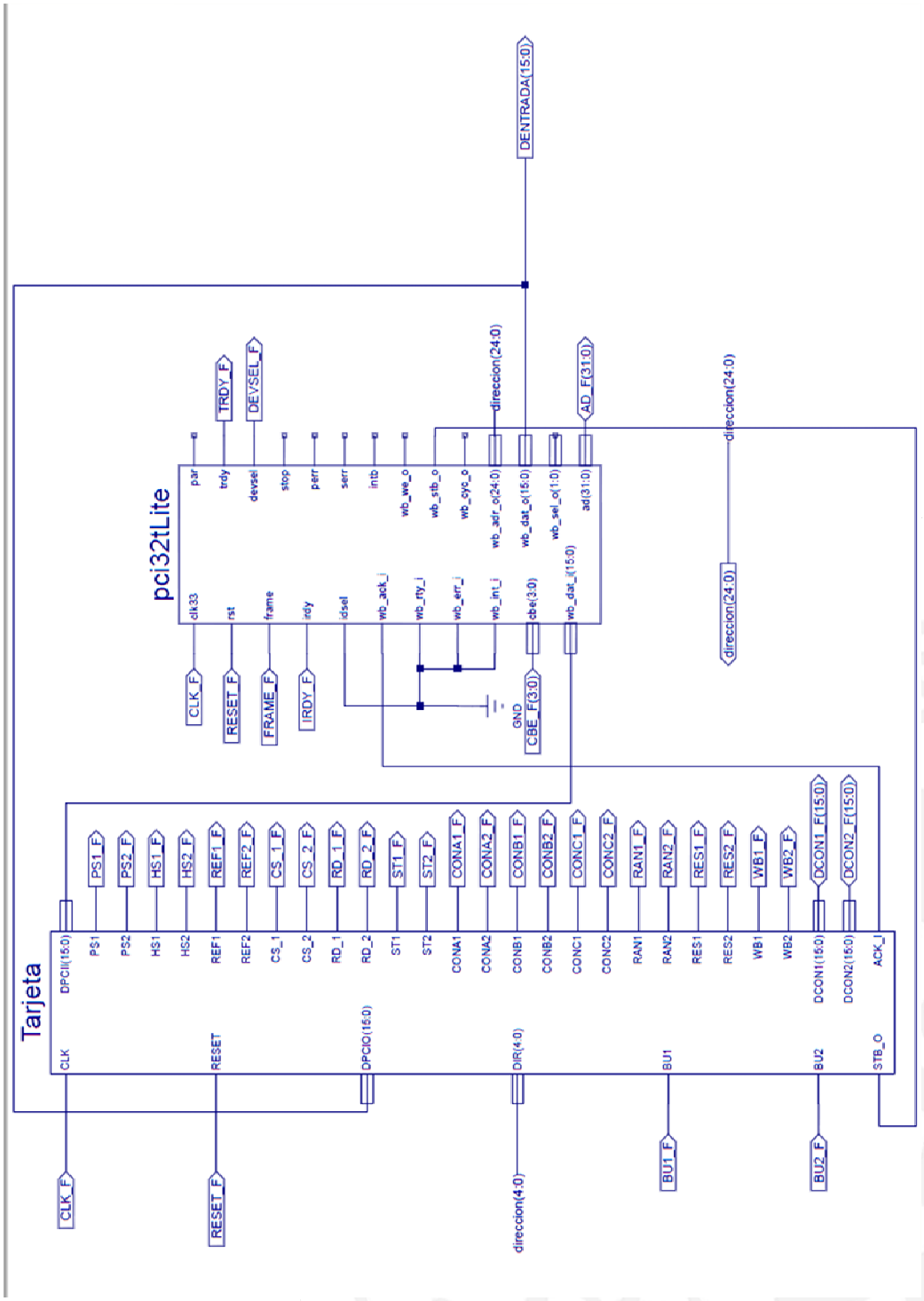


Parte III FPGA

Capítulo 1 TARJETA.SCH



Capítulo 2 TARJETAPCI.SCH



Índice del presupuesto

Parte I Presupuesto ***139***

Capítulo 1 Mediciones ***140***

1.1 Resistencias **140**

1.2 Condensadores **140**

1.3 Circuitos integrados **140**

1.4 Tarjetas **141**

1.5 Horas de ingeniería **141**

Capítulo 2 Precios unitarios ***142***

2.1 Resistencias **142**

2.2 Condensadores **142**

2.3 Circuitos integrados **142**

2.4 Tarjetas **143**

2.5 Horas de ingeniería **143**

Capítulo 3 Sumas parciales ***144***

3.1 Resistencias **144**

3.2 Condensadores **144**

3.3 Circuitos integrados **144**

3.4 Tarjetas **145**

3.5 Horas de ingeniería **145**

Capítulo 4 Presupuesto general ***146***

Parte I PRESUPUESTO

Capítulo 1 MEDICIONES

1.1 RESISTENCIAS

Valor	*Fabricante*	*Referencia*	*Cantidad*
1kΩ	VISHAY DRALORIC	CRCW06031K00FKEA	24
22Ω	VISHAY DRALORIC	CRCW060322R0FKEAHP	12

1.2 CONDENSADORES

Valor	*Fabricante*	*Referencia*	*Cantidad*
1nf	AVX	08052C102KAT2A	12
10uf	AVX	1206YC106KAT2A	12
0.1uf	AVX	06033C104JAT2A	20
1uf	AVX	0603YD105KAT2A	2
0.47uf	AVX	0603YD474KAT2A	2

1.3 CIRCUITOS INTEGRADOS

Tipo	*Fabricante*	*Referencia*	*Cantidad*
Operacional	TEXAS INSTRUMENTS	OPA2211AIDDA	6
Conversor	TEXAS INSTRUMENTS	ADS8556IPM	2

1.4 TARJETAS

Tipo	*Fabricante*	*Descripción*	*Cantidad*
PCB	Eurocircuits	PCB de 4 capas	1
FPGA	Raggedstone	Spartan-3 FPGA PCI	1

1.5 HORAS DE INGENIERÍA

El número total de horas de ingeniería para la realización de este proyecto ha sido de 1000 horas.

Capítulo 2 PRECIOS UNITARIOS

2.1 RESISTENCIAS

Valor	*Fabricante*	*Referencia*	*Precio*
1kΩ	VISHAY DRALORIC	CRCW06031K00FKEA	0.021€
22Ω	VISHAY DRALORIC	CRCW060322R0FKEAHP	0.043€

2.2 CONDENSADORES

Valor	*Fabricante*	*Referencia*	*Precio*
1nf	AVX	08052C102KAT2A	0.082€
10uf	AVX	1206YC106KAT2A	1.08€
0.1uf	AVX	06033C104JAT2A	0.034€
1uf	AVX	0603YD105KAT2A	0.156€
0.47uf	AVX	0603YD474KAT2A	0.197€

2.3 CIRCUITOS INTEGRADOS

Tipo	*Fabricante*	*Referencia*	*Precio*
Operacional	TEXAS INSTRUMENTS	OPA2211AIDDA	17.14€
Conversor	TEXAS INSTRUMENTS	ADS8556IPM	45.22€

2.4 TARJETAS

Tipo	*Fabricante*	*Descripción*	*Precio*
PCB	Eurocircuits	PCB de 4 capas	233.41€
FPGA	Raggedstone	Spartan-3 FPGA PCI	277.68€

2.5 HORAS DE INGENIERÍA

Se ha estimado un precio de la hora de ingeniería de 40€/hora

Capítulo 3 SUMAS PARCIALES

3.1 RESISTENCIAS

Valor	*Cantidad*	*Precio unitario*	*Total*
1kΩ	24	0.021€	0.504€
22Ω	12	0.043€	0.516€

3.2 CONDENSADORES

Valor	*Cantidad*	*Precio unitario*	*Total*
1nf	12	0.082€	0.984€
10uf	12	1.08€	12.96€
0.1uf	20	0.034€	0.68€
1uf	2	0.156€	0.312€
0.47uf	2	0.197€	0.394€

3.3 CIRCUITOS INTEGRADOS

Tipo	*Cantidad*	*Precio unitario*	*Total*
Operacional	6	17.14€	102.84€
Conversor	2	45.22€	90.44€

3.4 TARJETAS

Tipo	*Cantidad*	*Precio unitario*	*Total*
PCB	1	233.41€	233.41€
FPGA	1	277.68€	277.68€

3.5 HORAS DE INGENIERÍA

El coste total de las horas de ingeniería es de 40000€

Capítulo 4 PRESUPUESTO GENERAL

Concepto	*Precio*
Resistencias	1.02€
Condensadores	15.33€
Circuitos integrados	193.28€
Tarjetas	511.09€
Horas de ingeniería	40000€
Total	**40720.72€**